TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

LASSI NIEMISTÖ

VIRTUAL FIELDBUS – APPLICABILITY, TECHNOLOGIES AND EVALUATION

Master's thesis

Examiner: Professor Seppo Kuikka
Examiner and topic approved by the Faculty Council of the Faculty of Engineering Sciences on
8. May 2013

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Automation Technology
NIEMISTÖ, LASSI: Virtual fieldbus –
Applicability, technologies and evaluation
Master of Science Thesis, 65 pages
May 2013
Major: Automation Software Engineering
Examiner: Professor Seppo Kuikka
Keywords: Fieldbus, CAN, bus, virtualization, QEMU, VirtualBox, TCP, embedded, development, testing, communication

In the present-day software and automation development, different methods of virtualization have become popular, as the final hardware is then not required for software development. This allows earlier and faster software process, reduced time to market and more fluent workflow. As distributed automation systems generally rely on fieldbuses of various types, implementing a virtual and fully operational fieldbus is a necessity for efficient utilization of the virtualized system.

In this thesis, we take a comprehensive approach to virtual fieldbuses, from concept definition to experimental performance characteristics. We discuss the common behavior of different fieldbuses, list applications for virtual buses and compare possible implementation technologies such as TCP/IP and shared memory. Virtualization tools VirtualBox and QEMU are closely studied, as they bring additional challenges to data transfer.

From the practical point of view, the study presents our experiences on implementing a virtual CAN bus for embedded development. With an extensive set of features and platform support in our design, it demonstrates the utilization of multiple technologies. Using the virtual CAN implementation, we then show the measured performance characteristics and evaluate the solution against actual hardware.

Potential of virtual bus technology was proven by the performance measurements. Shared memory implementation provided extremely good performance, sufficient for implementing any virtual fieldbus system. It was also found to be efficient in respect to CPU load. Unfortunately, shared memory usually cannot cross virtualization boundaries. TCP was found as the best option for the rest of the use cases. In restricted local Ethernet or between VirtualBox and host OS, it is able to provide latencies under 700µs, similar to hardware performance. Observed bottlenecks were the use of the QEMU emulation tool without optimizations, and slow USB fieldbus adapters.

We recommend using virtual fieldbuses in virtualized development and debugging of distributed systems and for automatic system level testing, if timing requirements are not extremely strict. Remote virtual connection to a hardware fieldbus is also seen as a valid application. The technologies and adapters must still be carefully selected for best results.

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Automaatiotekniikan koulutusohjelma
NIEMISTÖ, LASSI: Virtuaalinen kenttäväylä -
Sovellettavuus, teknologiat ja arviointi
Diplomityö, 65 sivua
Toukokuu 2013
Pääaine: Automaation ohjelmistotekniikka
Tarkastaja: professori Seppo Kuikka
Avainsanat: Kenttäväylä, CAN, väylä, virtualisointi, QEMU, VirtualBox, TCP, sulautettu, kehitys, testaus, kommunikaatio

Virtualisointimenetelmät ovat suosittuja nykypäivän automaatio- ja ohjelmistotuotannossa, sillä niiden avulla ohjelmistoa voidaan kehittää ilman lopullista laitteistoa. Tällä tavoin on mahdollista nopeuttaa ja helpottaa ohjelmistoprosessia sekä aikaistaa tuotteen pääsyä markkinoille. Hajautetut automaatiojärjestelmät ovat yleisesti riippuvaisia erilaisista kenttäväylistä, joten esikuvaansa riittävästi vastaava virtuaalinen kenttäväylä saattaa olla ratkaiseva tekijä virtualisoinnin hyötykäytössä.

Tämä opinnäytetyö syventyy virtuaalisiin kenttäväyläratkaisuihin, konseptin määrittelystä suorituskykymittauksiin. Määritämme eri kenttäväylien yhteiset ominaisuudet, selvitämme virtualisoinnin mahdolliset sovelluskohteet ja teemme laajan teknologiakatsauksen kohdistuen muun muassa TCP/IP-yhteyksiin ja jaettuun muistiin. Käsittelemme tarkasti myös VirtualBox- ja QEMU-virtualisointityökaluja, joihin liittyy kommunikaatiota ajatellen tiettyjä lisähaasteita.

Teoreettisen ja teknisen selvityksen tuloksia hyödynnetään käytännössä toteuttamalla virtuaalinen CAN-väylä sulautetun ohjelmistokehityksen tarpeisiin. Toteutuksen laajat ominaisuudet sekä tuki useille alustoille mahdollistavat useiden teknologioiden kokeilun. Väylää käyttäen tehdään lukuisia suorituskykymittauksia, joiden avulla suoritetaan vertailu eri tekniikoiden välillä ja suhteessa laitteistopohjaiseen väylään.

Tulokset todistavat virtuaalisen kenttäväylätekniikan potentiaalin. Jaettua muistia käyttäen vaativimmatkin sovellukset ovat mahdollisia, prosessorikuormituksen jäädessä samalla vähäiseksi. Jaettu muisti ei kuitenkaan usein sovellu virtualisointirajoja läpäiseviin yhteyksiin. TCP-yhteyden katsottiin soveltuvan muihin käyttötapauksiin parhaiten, sillä sekin mahdollistaa lähes laitteistotasoa vastaavat alle 700 µs viiveet VirtualBox- ja paikallisverkkoympäristössä. Eniten suorituskykyä alentavat QEMU-emulaattorin käyttö ilman optimointimenetelmiä ja hitaan USB-kenttäväyläadapterin liittäminen järjestelmään.

Suosittelemme virtuaalista kenttäväylää käytettäväksi hajautettujen järjestelmien ohjelmistokehityksessä, ongelmanselvityksessä ja testauksessa, olettaen että ajoitusvaatimukset eivät ole äärimmäisen tiukat. Etäyhteyden muodostaminen kenttäväylään virtuaalitekniikoiden avulla on myös toimiva sovellus. Toteutustekniikka ja laitteet on kuitenkin valittava huolella parhaiden tulosten saavuttamiseksi.

PREFACE

This is a master's thesis written for the Department of Automation Science and Engineering in Tampere University of Technology. With this publication I wish to provide both scientific and practical information about virtual fieldbus technology for embedded software and automation industry. A sincere commendation belongs to Wärtsilä for funding this research and to Jari Kuusisto (Wapice Oy) for the initial thesis subject arrangements.

I want to express my special thanks to both of my personal thesis mentors, Prof. Seppo Kuikka (TUT) and DSc. Sakari Junnila (Wapice Oy), for the valuable support and comments during the writing process. Your honest interest in the subject gave good motivation also for me. I am also very grateful for the essential assistance received from my colleagues Jaakko Karrenpalo and Kåre Särs (Wapice Oy) in solving technical problems. I want to thank also my employer Wapice Oy as a company, not only for providing the facilities, tools and support needed for the research, but also for the recognition of my scientific work.

Besides the above mentioned funding and research cooperation, the support of my wife, friends and family has been crucial. The thesis process has often required working late hours, but I have always received the understanding and encouragement to keep moving forwards. Thank you all!

Tampere 21.5.2013

Lassi Niemistö
lassi.niemisto@iki.fi

Table of contents

LIST OF TERMS AND ABBREVIATIONS

Fieldbus tunneling	Connecting two or more hardware fieldbuses or clients together by routing fieldbus frames over computer network.
Hub	The central component of our virtual CAN implementation. Hub accepts client connections and implements the bus logic.
IPC	Inter-process communication. Data transfer between the processes executed in parallel on the same computer.
KVM	Kernel-based Virtual Machine. A full virtualization solution for Linux on x86 hardware containing virtualization extensions.
LowLevelCAN	A library for abstract connectivity to hardware CAN adapters or virtual CAN bus. Used in our example solution.
Nagle	Algorithm for collecting TCP data into larger physical frames in order to reduce overhead.
Virtualize	Emulate behavior of target hardware in order to run embedded software on a PC.
Paravirtualization	Virtual machine is conscious about the virtualization and may thus use lightweight virtual devices instead of complete hardware emulation.
QEMU	Generic and open source machine emulator and virtualizer. Supports also other than x86-based platforms.
RTT	Round trip time. A delay measured between sending a message to a communication endpoint and receiving an immediate reply from it.
SocketCAN	A set of open source CAN drivers and a networking stack. Supported by the Linux kernel.
TCP_NODELAY	An option for TCP protocol to disable the Nagle algorithm and send messages immediately.
Transport protocol	A protocol operating on the transport layer of the OSI model. Provides end-to-end transmission for applications. For example TCP or UDP.
VirtualBox	General-purpose full virtualizer for x86 hardware, targeted at server, desktop and embedded use.

1 INTRODUCTION

Digital fieldbus technology has achieved a strong role in automation and control industry, allowing the computation to be distributed into a network of field devices instead of a having a central controller. When considering the amount of needed wiring, data integrity and diagnostic features, the benefits of digital communications are obvious. However, the binary form of data transportation requires a vast amount of logical processing both when sending data and receiving it. Fieldbus communication logic related engineering demand does not apply on the bus hardware industry only. It has brought protocol programming as a part of almost every embedded software project, not to mention the development of all the indirectly communication-based functionality.

The above-mentioned transition from centralized control to distributed field device networks has also led to greater variety of programmable devices with different hardware involved in equal scale systems. Besides, the hardware itself is often designed exclusively for each project. For logistic and economic reasons, the actual hardware might not be available for every developer in the project team. Using the actual device for all debugging and testing purposes is also often impractical. As an alternative, it is possible to run the embedded software virtualized on a PC computer, allowing many development tasks to be carried out without the target hardware.

Contrary to executing the software code itself, virtualizing its connections to environment is rarely trivial and thus often left out of concept. While lack of actual sensor measurements or controllable outputs is seldom critical, the absence of communication effectively paralyzes significant amount of the intended features. Implementing a virtual, yet fully operational fieldbus may thus become a necessity for efficient utilization of a virtualized system.

Some applied research on the subject has already been performed by Jong-Seo, Sang-Hun and Hyun-Wook, who implemented a virtual CAN bus for modular avionics [1]. Also McLoughlin has used a virtual fieldbus for development and testing purposes [2] and Obermaisser and Peti implemented a virtual CAN bus as a subservice of a safety critical time triggered protocol [3]. However, no focused study or a general solution was found, which created the need for such research to be performed for Wärtsilä. In this thesis, we will take a comprehensive approach to the topic, discussing it both theoretically, practically by presenting an example solution and in the light of measurements. Rather than focusing on our implementation in detail, we will keep the discussion on a generalizable level to be of interest to larger group of readers.

1.1 Thesis outline and objectives

The potential of virtual fieldbus technology seems promising, but only little practical information for adoption has been published. Thus, we state the following research questions: In which *areas* virtual fieldbus technology is successfully applicable? How *close* to hardware performance it is possible to get with a virtual solution? What implementation *technologies* will give the best results? The subject will be discussed primarily from the embedded software development viewpoint.

The thesis is started with an introduction to virtual fieldbus concept, also examining performance metrics and existing solutions. In addition to gathering this essential information for later chapters, the goal of the theoretical consideration is to find out the benefits achieved by using a virtual fieldbus, not forgetting the possible challenges and drawbacks. We do not restrict the discussion to any particular fieldbus type or standard, although CAN bus is mostly used for examples. As in the entire thesis, we assume the reader to have an engineer level understanding of fieldbus technology. Also, basic knowledge of embedded programming, electronics and computer networks is required to perfectly follow the text. These assumptions are made to stay within reasonable scope for a thesis, and to ensure concentration on the actual interests.

We will then continue to a technology study, which evaluates different implementation alternatives for virtual fieldbuses. To produce scientifically valuable information, the text aims at widely finding and arguing the benefits and disadvantages of each detected solution. Its results are then applied to a design and implementation of a virtual CAN bus system. Beneficially, the large feature set of this particular project provides a prolific basis for prototyping several technologies. The constructed solution is explained in adequate precision, excluding the small implementation details to focus on relevant discussion.

To provide answers to the question about virtual bus performance with different technologies, appropriate measurements are conducted using the metrics discovered in the theory part. The experiments are documented in two sections, one describing the test setup and methods, and the other presenting the results. After going through the above described four-part research process, all the findings are summarized and discussed in a designated conclusion chapter. The stated thesis questions are to be answered distinctly and the successfulness of thesis process is also evaluated.

2 VIRTUAL FIELDBUSES

In this chapter, we will discuss fieldbuses and their virtualization from a theoretical and general viewpoint. This will be started by an introduction to the virtual fieldbus concept, followed by describing its essential applications and listing the existing products and solutions found. Using this initial information, we can then spot the important features associated with a multi-purpose fieldbus virtualization, and also find the challenges and drawbacks involved. We also look for and discuss metrics to be used in measuring virtual bus quality and performance. As a result, we can then constitute reasonable expectations for a virtual fieldbus solution. The gathered knowledge will be used in the following chapters as reference for design, as baseline for validation and ultimately to identify the applicable uses for virtual bus technology.

2.1 The concept

To clarify the forthcoming discussion, we need to define what is denoted by virtual fieldbus. This, naturally, is based on the concept of a fieldbus itself. Defined by the comprehensive fieldbus standard series IEC 61158, a fieldbus is a bidirectional serial digital data transmission bus, which allows the communication among industrial information devices such as sensors, actuators and controllers [4, see 5 p. 18]. The standard set includes specifications all the way from lowest level definitions, elements and different communication layers to a selection of individual protocols [6]. Being unfortunately not available for this research in full-text, their content still gives a good overview of the fieldbus concept, spreading over all the OSI model layers.

Virtualization, generally, means transforming something to a computer-generated simulation of reality [7]. When considering virtualization inside the computer world, hardware is practically the only element comparable to physical reality, which focuses the definition of virtualization to simulating hardware in software. Fieldbuses make no exception to this, as their functionality outside the hardware devices is essentially implemented in software. Therefore, we define virtual fieldbus in general as a fieldbus using virtual hardware as its transmission media, with software-based and logical parts kept as original as possible to preserve its features.

2.1.1 Virtual fieldbus and OSI model

While virtual fieldbuses are clearly a special case of fieldbuses, we are primarily interested in the differences involved. As stated above, any higher software layers, such as CanOpen protocol stack and Modbus application layer [8, ch. 31.6, 36.4], are not to be modified in the virtualization process, and thus left out of our virtual fieldbus concept. Referencing to OSI model [8, ch. 1.3-1.5], the highest responsibility of the bus is thus to provide end-to-end connections on the transport layer.

Starting from the lower end of the OSI model, the technology utilized on the physical layer varies greatly among the bus standards and brands. Profibus and CAN, for example, use twisted-pair cabling [9, p. 32; 10, p. 5], while LonWorks can also communicate using power lines, fiber optics and wireless methods such as radio and infrared [11, p. 31]. Our viewpoint to the bus virtualization is still more on the application side, making all the details on the physical layer uninteresting, if they do not have effect on the higher layers. While seeking general purpose solutions and results rather than specializing in any single fieldbus type, we also practically have to rule out the physical layer. The overall picture can now be seen from Figure 1.

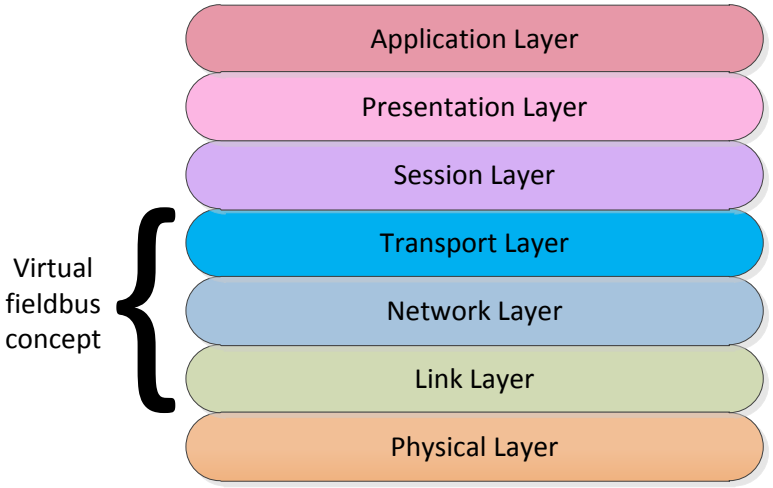

***Figure 1**: Virtual fieldbus scope in OSI model*

2.1.2 Communication characteristics

The domain of a virtual fieldbus is now limited between the link and transport layers, but further qualification is still needed for a generalizable concept. On these layers, differences among the fieldbus types mainly associate to performance values and rules of the transmission logic. At this point, we can identify the growing set of Ethernet-based fieldbuses, such as EtherCAT [12] and PROFINET [13], and exclude them from upcoming discussion, because Ethernet already has well-developed virtualization methods [14; 15]. In the remaining group of bus types, more deterministic and simple serial

communication seems to dominate. Characteristics for a group of typical fieldbuses are given in Table 1.

***Table 1**: Characteristics for a selection of fieldbus types*

Fieldbus type	Frame size (bytes)	Throughput	Arbitration	Reference
CAN	13	10..1000kbps *	Collision detection, priorities	[16, p. 21, 42-55; 17]
FOUNDATION Fieldbus H1	258	31.25 kbps	Active scheduler	[18, p. 355, 359]
INTERBUS	512 *	500 kbps - 2Mbps	Master/slave, summation frame	[8, ch. 33.1-33.9]
Modbus	256	1..100 kbps *	Master/slave	[19, p. 5; 20]
PROFIBUS	249	31.25 kbps - 12 Mbps	Token passing	[9, p. 124; 18 p. 350, 351]
LonWorks	250	5..1250 kbps	CSMA	[8, ch. 41.3; 11, p. 41, 45-46]
*Not strictly specified				

The comparison clearly shows, that all the bus types share a common data transfer method, a message frame with restricted size. The frame size varies between 13 and 512 bytes, being remarkably small in the gigabyte scale of modern computing and also smaller than for example the standard 1500 byte Ethernet frames [21]. The variance in throughput is higher, but it occurs equally both within one bus type and among the whole group. Arbitration methods depend on the media and topology utilized, and in some cases additional signaling or tokens are required. The purpose of arbitration is still, independent of the bus type, to determine strict frame transmission order based on its characteristic logic. Most fieldbus types as CAN, Profibus and Modbus, provide automatic data corruption detection and retransmission [16, p. 23; 8, ch. 32.5, 36.11], and consequently also the virtual fieldbus, in general, must guarantee data integrity.

On the physical layer, real fieldbus timings are deterministic, but this does not necessarily mean application-level timings to have the same accuracy. A fieldbus type may support multiple media types with different timing properties. Arbitration may alter transmission times by prioritizing certain messages, and in an overload situation some messages may even be dropped. In addition, all added hardware and software components will cause extra delay before frames reach the application layer. Probably for these reasons, official response time specifications were not found for the examined fieldbuses. With many fieldbus types, sub-millisecond class delays are considered advanced performance [22; 23]. From the reported values found, lowest round-trip latencies of 250-500 µs are associated with CAN bus [1, p. 3-4]. Generally said, fieldbus features and applications vary from non-real-time to hard real-time. Based on the discussion this far, Table 2 defines the communications properties of our virtual fieldbus concept.

***Table 2**: Fieldbus communication characteristic features*

Frame size	≤ 1 kb
Throughput	≤ 15 Mbps
Frame order	Strictly defined by characteristic rules
Round-trip time	250µs at shortest, 1.0ms sufficient in most cases
Data integrity	Guaranteed in normal conditions

2.2 Applications for virtual bus technology

In embedded software development, the requirement to run the software on the actual target device may result in serious inconveniences. Fully working prototype hardware may not be available for the development team, and even if it is, the debugging features may be limited and workflow may be decelerated by the repetitive demand of reprogramming the flash memory devices [2, p. 20-22]. For these reasons, the usage of different virtualization methods for running the software on PC has become popular, as they allow earlier development, reduced time to market and more fluent workflow [24, p. 6]. However, heavily fieldbus communication dependent projects require virtualizing also the bus communications to enable development and testing of the fieldbus related parts [25, p. 14].

As stated, virtual fieldbus is a great aid for development-time testing. Its possibilities are still not restricted to manual workflow. Automated testing frameworks already exist also for embedded systems, National Instruments TestStand [26] being a good example, but they naturally require heavy instrumentation as the tests are run on actual hardware. Also a more traditional test automation system, as Jenkins [27, p. 135-136], may be used to test embedded systems through their external interfaces. Without a virtualized fieldbus, communication-dependent testing is in both cases possible only with a pre-constructed hardware system. It is thus not difficult to picture all the improvements that can be achieved with a virtualized bus system. In a completely virtual environment, every test script is able to set up its own bus configuration and even alter it during the test run. More sophisticated bus implementations could allow simulating different bus error situations, controlled by the test script. This surely enables writing better and faster test cases, not to forget decreased hardware and instrumentation costs.

Physical properties of hardware buses effectively restrict the acceptable cabling lengths, and setting up cables between distant locations is usually tedious or impossible, at least for temporary purposes. Wired Ethernet or other IP networks are available practically everywhere, not to mention the wireless solutions, so utilizing them as a tunnel between two fieldbuses is absolutely beneficial. This method might be questioned of not being a virtual fieldbus, as hardware network is used as data carrier, but the virtualization software layer is still needed for emulating the characteristic features of the bus. In

addition to the mentioned hardware bus interconnections, network tunneling technology adds flexibility also to partly and fully virtual fieldbuses. The communicating virtual devices no longer have to be located on the same computer, but one may for instance run them distributed on a set of high-performance virtualization servers and development workstations, still retaining seamless communication.

Although most of the applications for virtual fieldbus technology exist in the field of development, a bus virtualization can of course be part of the final product as well. In modular software architectures, it may be useful to create a virtual bus for the software units running on a common platform and thus sharing a single fieldbus connection. From a single software module point of view, the other modules executed on the same platform are invisible and the virtual bus appears as exclusive access to the hardware bus connection. [1, p. 1-2] As a drawback, applications of this type certainly have more rigorous requirements for the implementation performance and stability, as the virtual bus will be in continuous use on the field.

2.3 Existing solutions

Before starting a new design and implementation for a virtual fieldbus, the existing alternatives should be investigated, firstly to verify the necessity of a custom solution, and secondly to gather ideas and perspective for the design. When considering the above listed applications, fieldbus-over-Ethernet gateways seem to be most popular in the range of commercial products. These devices are provided by multiple manufacturers, especially for CAN bus. Three typical solutions, VScom NET-CAN 110 [28], esd electronics EtherCAN/2 [29] and SYS TEC CAN-Ethernet Gateway GW-003 [30] are pictured in Figure 2. They provide also drivers and application programming libraries for connecting PC computers or custom devices to the bus system [31; 32; 33]. The versatility of the software parts varies, but none of the examined APIs seem to offer pure virtual utilization, which is of course natural from the marketing point of view.

***Figure 2**: Three commercial Ethernet-CAN gateway solutions [28; 29; 30]*

One existing way to create a simple virtual bus is to use a virtual device driver with multi-access support, simply shared among the virtual fieldbus clients. Kvaser, for instance, offers this option in their USB CAN adapter library [34]. This technique is still possibly not designed for continuous and extensive use. Using a device driver as a vir-

tualization base may provide connectivity to the corresponding hardware device, but this is likely to be limited to a single device at a time.

The last existing solution to introduce is much less device-oriented. Multi-purpose modeling and simulation environments are a growing trend also in the embedded development [35, p. 1-3]. In addition to aided design and testing, these tools can even produce application code for the end product [36]. Among all the interesting features, the environments can be used for bus virtualization and connections to fieldbus hardware [37; 38, p. 9]. A Simulink-based example of virtual CAN bus is presented in Figure 3. This type of bus virtualization has the major benefit of modifying and routing the bus traffic on-the-fly, enabled by the advanced simulation features. Also, extension capabilities are usually provided through programming interfaces. Despite of the advantages, this solution is practical only when developing is model-based, because using a full simulation environment only for the fieldbus might be inordinate.

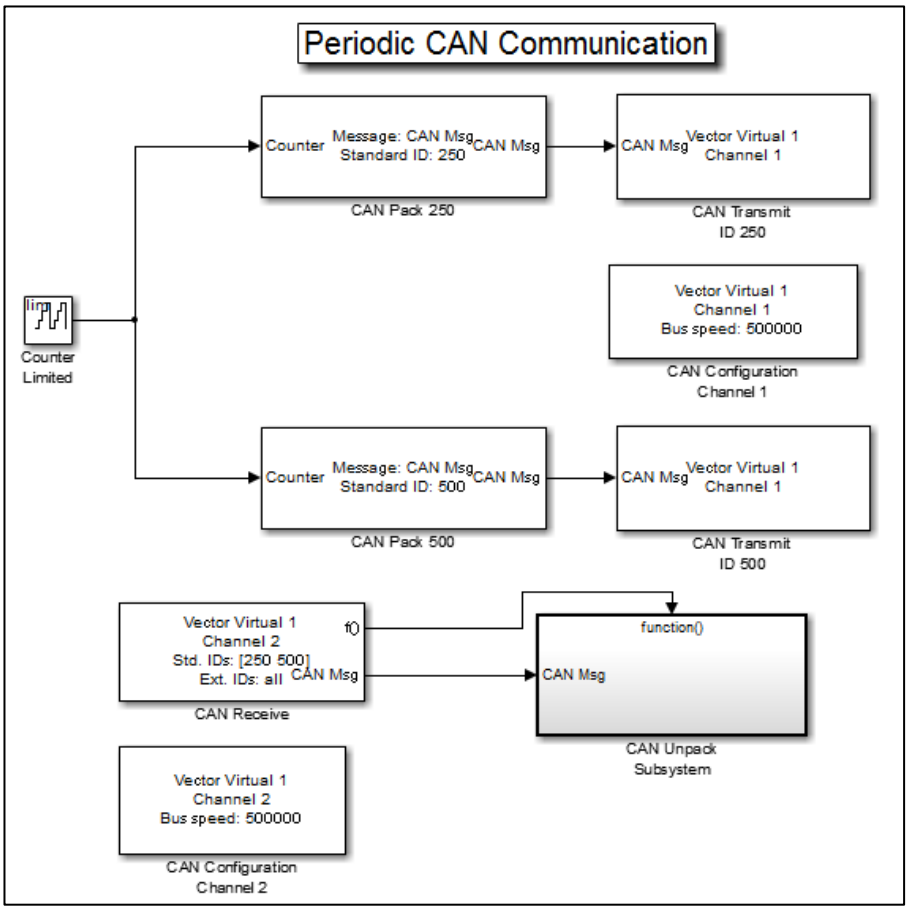

***Figure 3**:A Simulink-based CAN bus virtualization example [39]*

2.4 Important features

The theoretical concept of a virtual fieldbus together with its discovered applications, both existing and prospective, still does not form an articulate enough base for implementation discussion. This chapter intends to distinguish and categorize the general and characteristic requirements involved in a virtual fieldbus design.

As stated above, the PC-fieldbus adapter manufacturers often provide fairly versatile drivers, programming interfaces and fieldbus related tools, but unluckily they seem to be often limited to be used only with the manufacturers' own hardware [40, p. 8; 41]. Some successful development has been carried out to create more unified and open interfaces, mostly in the Linux world. For example the universal CAN driver LinCAN [42] has a wide support for different bus controller chips, the socket based CAN library SocketCAN has achieved the role of official Linux CAN driver interface [43], and the VCA library acts as a common interface for the two alternatives [44]. As the existing libraries also provide some virtualization possibilities, an adequate virtual fieldbus must provide decent hardware support to compete them. The mentioned widely adopted interfaces may also be beneficial if supported by the virtual bus, as they may allow instant connectivity to a set of existing applications and devices.

The virtual fieldbus concept sets requirements for the communication methods to utilize. First, the virtual bus must guarantee data integrity, as hardware bus controllers do in most cases. Second, the virtual bus must keep the frame order strictly as defined for the emulated bus type. Both cases also naturally require lossless transmission. If such a reliable technology is not available, these features must be provided by the software wrapping the unreliable connection. Multiple separate channels are commonplace in real world applications for redundancy or segmentation, and thus should be supported in the virtual implementation also. Beside these logical properties, the communication performance has a crucial role. As the needed throughput varies among the bus types (Table 1) and the real-time requirements depend on the application, we can only point out that low latency and high throughput increase the universality of the solution. In many cases, the throughput is to be artificially limited to the emulated bus speed. Inside a local system the communication methods can be selected quite freely, but support for network tunneling may be required directly as a feature.

Another category of features is formed by the bus control. A virtual bus setup always needs to be configured, analogous to wiring hardware buses between device ports. As the wiring is highly application dependent, the virtual implementation should not place restrictions on this. For testing purposes, it is ordinary to modify the bus configuration and disconnect, cut or short circuit cables when devices are running. This should be possible also in the virtualization to enable full-scale virtual testing. In addition to these hardware derived features, the virtual bus enables creating also another valuable testing aid, not entirely available by other means: it should be possible to discard, modify or corrupt fieldbus frames on-the-fly randomly or based on filter conditions. While all the above pictured events are outside of the bus client's control, a mechanism and interfaces for configuration and control must be built in the virtual bus system itself. Interfaces may be required both towards user and for other programs, and in a distributed bus system it is conventional to have control access from any of the interconnected locations.

While the above described features already guarantee a multi-functional solution, its utilization may still be limited by the platform. Several platform types may be con-

nected using Ethernet tunneling, and virtualization or emulation is often used to create independent sub-environments, both parallel and nested. Because of this, different platforms often exist also inside a single bus setup, which makes portability even more important for virtual fieldbuses compared to many other types of software. In addition to regular Linux or Windows based workstation systems, also more plain embedded environments should be supported. This is primarily needed to allow connecting virtualized embedded devices on the bus, but also enables implementing for example gateway devices on real hardware.

2.5 Challenges and drawbacks

By now we have discussed the virtual fieldbus from a conceptual viewpoint and formed a picture of an optimal solution. Every software project aims at filling its requirements as well as being feasible, but some demands are more challenging than others and some even impossible to implement. In this section, we will discuss the characteristic challenges in virtual fieldbus design and implementation and look for probable disadvantages remaining even in a most successful design.

Network tunneling has been listed above as an important feature for distributed bus setups. It is also one of the most obvious threats to the bus communication quality, because the widely used Ethernet standard IEEE 802.3 does not support any real-time guarantees [45, p. 1]. If the network is not dedicated for the fieldbus, other users may also cause occasional overload. Such variance in quality is more harmful from the latency point of view, compared to average throughput. While even Internet connections these days usually achieve enough bandwidth for most of the fieldbus types, their multi-millisecond delays are far from typical fieldbus timing properties. The delays drop significantly when restricting the communication to wired high speed local area networks, and even more when operating in a virtual network, supported by all major operating systems. Timing issues can be thus reduced by selecting the network components and topology carefully, but major part of flexibility is lost simultaneously. As a result, even perfect support for network tunneling itself clearly does not guarantee applicability in all combinations of fieldbus requirements and network types.

In the embedded software development, the target device architecture is seldom the same as in development workstations. To enable more realistic virtual testing of the software, the actual processor of the device must be emulated [2, p. 20]. For this, there are several tool alternatives, including SkyEye and ARMware, but the open source machine emulator QEMU has become a popular option [24, p. 3; 2, p. 22]. Emulation may naturally result in poor performance, and even multiple decades slower execution compared to native code has been reported in an extreme use case [46, p. 5]. Though the reduced performance is mainly the problem of virtualization in general, it needs to be taken into account in the virtual fieldbus design as well, especially because the fieldbus setup often consists of multiple virtualized devices. In some cases, this might require a dedicated high performance server for running the virtualized system. As in networking,

latency is still more important than throughput, here referring to computing power. While embedded devices tend to use a real-time operating system or no OS at all to fulfill the timing requirements [47, p. 10], workstation computers normally use asynchronous operating systems instead. Therefore, emulating or virtualizing the former system type on top of the latter may add unpredictability also to fieldbus communication.

While setting up the system emulation using the above-mentioned tools is rather simple, methods for setting up fieldbus communication to and from the emulated virtual machine may be found challenging. In fact, the similar problem turns up in any system, where data transfer must cross virtualization borders. [48, p. 1-2] One such use case is when virtual machines are used to build a development environment. When parallel virtualized guests exist, their inter-communication may even need to always pass through the host, doubling the problem [48, p. 2]. In many other cases file share [49, p. 66], virtual networking [15] and other well-supported techniques can be used to provide sufficient data transfer, but the timing requirements of virtual fieldbus may rule most of them out. In addition, high performance communication technologies seem to be mainly developed for the needs of server oriented virtual machine hypervisors, such as Xen and VMware [48, p. 27], and may thus not be supported when emulating more obsolete platforms used for embedded devices.

The virtual bus cannot always retain the natural topology of a hardware bus. To implement the characteristic fieldbus logic with strictly defined frame order, a shared central element is often needed, analogous to a common bus cable. For example linear bus type topology thus becomes actually a star. Such topology conversion does not need to be disadvantageous, but it might be challenging to design. In extreme cases, connections to hardware buses or network tunneling can lead to multiple star centers and thus make the desired bus model even impossible to implement. A complicated topology or logic might also force the frames to pass through multiple message buffers, causing extra delays to communication.

2.6 Metrics

To be able to compare and evaluate virtual bus implementation technology selections and complete implementations, suitable metrics must be defined. When it comes to the physical layer of a hardware fieldbus, the measurable quantities as voltage, resistance and edge quality [50] are numerous, and naturally dependent on the media type used. Luckily, the virtualization concept generalizes the media to simplified frame transfer, where the characteristic values are much fewer. Due to the technical and similarity, the well-studied measurement theory and methods from the Ethernet field [51; 52] may be applied to virtual fieldbuses as well. When Ethernet or IP networks are explicitly or implicitly used for virtual fieldbus data transfer, a wide set of existing benchmark tools is available, including netperf [53], iperf [54] and ping. In the following subsections, the different performance and quality attributes are identified and discussed.

2.6.1 Data rate

The most obvious subject to measurements is probably the bus data rate, because in hardware solutions it is often a timing-based constant, but varies in the virtual environment. When discussing data rate, the terms throughput and bandwidth are often considered synonymous [55, p. 1]. However, they have slightly different meaning: while bandwidth represents the maximum data transfer capability of the media regardless of the content, throughput instead is defined as the successful message delivery over the communication channel within a certain communication concept [56]. Because the purpose of fieldbus is to provide transport for any application data, bandwidth is a more relevant term, when measuring its data rate in general. Virtualization brings another aspect to this, as other communication technologies are used as data carrier and the virtual fieldbus thus acting as an application with relevant concept of throughput. Depending on the delays involved and protocols used, the virtual bus throughput may not fully utilize the carrier channel bandwidth [56]. When measuring the fieldbus performance it makes no difference which term we use, but the conceptual differences and the related problems must be taken into account in evaluating different data transfer technologies for the virtual implementation.

Regardless of the data rate meaning, measuring it is technically similar. As a unit, any data packet in unit of time may be used, but bytes per second would be the most comparable when the frame size varies. By definition, data rate is measured as an average from a selected time period [57, p.1]. When measuring throughput or bandwidth, the data rate is to be measured under maximum bus load. For this, a suitable test setup producing artificial traffic must be constructed. Because virtual fieldbus should guarantee a constant bandwidth, we are mainly interested in two values: the long-time average and the variance of short time average. Throughput may also depend on the packet size if overhead processing becomes the limiting factor [57, p.5]. Thus, the throughput should be measured also with multiple frame sizes.

2.6.2 Delay

In addition to the bandwidth, transmission delay is an import attribute, when considering systems with timing requirements. By delay, we mean the time elapsed while a message is sent and received between two points in the communication system. Unlike throughput, delay is not essentially an average value, but associates with an individual frame. When asynchronous communication methods are used, the delay does not stay constant, but instead fluctuation, often called jitter [58], is experienced. In a hard real time system, data frame loses its value completely if certain reception deadline is not met. When the system is soft real time, the value of data is not completely lost but still decreases in time. [59, p. 1] For these reasons, we find average delay not very interesting. Instead, the distribution of the delay, or more specifically the probability of the delay being lower than a selected deadline can tell us more about the communication suitability for an application.

If the application and its tolerable latencies are known, scalar probability values are informative, but for an overall picture it is best to present the delay distribution graphically. Measuring one-way transmission delay directly would require the sender and receiver to have precisely synchronized clocks, which is not always possible. Instead, a more popular method seems to be measuring a round-trip time (RTT), by requesting an immediate answer from another communication client and using only the sender clock for calculating the time elapsed in these two transmissions. However, one-way delay can be reliably calculated from RTT only in perfectly symmetric systems. Total delay is the sum of delays originating from different components of the communication system, including both hardware and software. [60, p. 2] To approximate the influence of each component, it may not be enough to perform the measurements on the application layer clients only, and thus adding measurement points to the lower layers may be needed.

2.6.3 Other measures

Transmitting messages over a communication channel needs to comply with a protocol, because data is generally only a stream of bits on the physical layer. It usually requires extra information to be transmitted with the actual message, defining for example type, length and error correction values. This so-called overhead is increased on every layer of encapsulation. [61, p. 1] Depending on the protocols used, the overhead may consume a significant amount of the total required bandwidth, even more than the actual payload data itself if small packets are used [62]. Opposed to most characteristic quantities, overhead magnitude may be calculated theoretically by analyzing the protocols. Most informative representation is obviously the percentage of overhead included in the total traffic. This value is, however, not a constant, but clearly a function of payload size and message type. When designing a virtual fieldbus, we are not interested in the overhead in the fieldbus internal traffic, but instead the one of encapsulating that traffic for virtual transportation.

A hardware communication media often suffers from noise and disturbance, and data may thus get corrupted or even lost during transmission [63]. The severity of the phenomenon can be reported by calculating the error and frame loss probabilities using generated traffic. As stated previously, a fieldbus usually provides error detection, correction and/or retransmission to cope with these issues, and a virtual alternative should thus also provide essentially error free transmission. Using the right technology, this should not become a problem in virtual environment, where data integrity is a requirement anyway. Consequently, as both of the probabilities should not exceed zero in a normal situation, evaluating them should belong to basic functional testing of the virtual bus rather than being part of the performance or quality measures. On the other hand, the values may become meaningful, if controlled artificial corruption or loss is part of the bus functionality.

In addition to all the metrics related to the communication performance and quality, it is also possible to measure its impact to the surrounding environment. This is realized through the resource consumption of the virtual bus system, mainly including the CPU

time, memory utilization and hardware communication device usage. From the three, memory consumption can be neglected, as buffering thousands of fieldbus frames only would require megabytes, a very low demand in a today's computer system. Communication devices lose certain bandwidth for the virtual fieldbus, which obviously restricts other uses of the shared media. Measuring and approximating this brings us back to the concepts of data rate and overhead, and thus no special methods are needed. CPU utilization requires attention, as we already found it a possible bottleneck in the virtual environment. Luckily, CPU usage statistics are usually provided by the operating system [64, ch. 6.4; 65] and it is relatively simple to compare the values with and without the virtual fieldbus running. Like data rate, it is an average value by definition, and practically always reported as a percentage.

2.7 Reasonable expectations

Before moving on to more practical parts of the thesis, it is time to form a conclusion on the findings of this chapter. Despite the various fieldbus types involved, it was possible and rather straightforward to form a generalized concept of virtual fieldbus. This justifies the rest of the research, which is thus applicable not only on a special case, but generally on its domain. Many applications and arguments for a virtual fieldbus were discovered, but directly related scientific publications were found significantly rare. The examined commercial or other existing solutions also seem not to suit all virtual bus applications, as their main target group was slightly different. It is thus firmly expected, that implementing a better solution with support for larger set of important features would be well possible.

In addition to prospects, we found also challenges. All communication methods will not likely provide low enough latency in fieldbus use. A deeper study on different alternatives is needed and will be conducted in the next chapter, but the existing publications give us a clue what to expect. As defined above, 250-1000 µs round-trip delays are enough for virtual fieldbus. Multiple inter-process communication methods have been reported to provide one-way latencies smaller than 50 µs [66, p. 6-10]. Also, a published virtual CAN solution only induced 8-60% of additional latency when extending a hardware bus [1]. Based on this, we assume that satisfactory timing properties can be achieved at least in restricted virtual environments.

3 IMPLEMENTATION TECHNOLOGIES

During the theoretical discussion, we have simultaneously formed an informal requirements specification and a brief risk analysis, a valid starting point for a generalized virtual fieldbus project. We will now continue on the same path of a software process, proceeding to the architecture and technology design consideration. Finding and selecting the right implementation technologies plays very strong role in filling the requirements. As our interest is also scientific, we target at evaluating different technologies widely and thoroughly, not only for the needs of the example solution presented later in this thesis. In the following chapters, technologies for the main functionalities are discussed, highlighting the areas where biggest challenges were seen.

3.1 Network communication

To support distributed virtual fieldbus access from different PCs and devices involved, communication over network is required. Computer networks are built on various wired and wireless medias, Ethernet being the most common in local area networks. However, in some virtual fieldbus applications, also wireless 802.11, broadband Internet or even mobile communications are useful. Different physical layers obviously have different timing properties, and it is thus important to distinguish between use cases. Only fast networks can be used for extending virtual fieldbus at hardware-like performance, while almost any network will connect remote clients with justifiably higher latencies. On the link layer, the network technologies have different protocols and frame formats, but the widely supported Internet Protocol family (IP) provides a common facade to all of them.

The real-time properties of computer networks were already found problematic in the previous section. In the field of Ethernet, lots of effort has been put in developing methods for better applicability in real-time applications. Some of them provide even hard real-time guarantees in switched Ethernet, but special real-time layers in each component of the network or even custom devices are needed [67]. Due to these restrictions, such technologies cannot be used in any networks not explicitly built for this purpose. Luckily, also the real-time extensions are often built under IP layer, and the common interface towards virtual fieldbus is thus retained regardless of the underlying technology. We can thus safely select the upper layers based on IP.

3.1.1 Transport protocol selection

Building fieldbus communication over IP still has multiple options for transport layer protocol. For continuous data transfer, the most common alternatives are the connec-

tion-oriented TCP (transmission control protocol) and the connectionless UDP (user datagram protocol). At first glance, we notice that TCP provides exactly the desired main features for communication: data integrity, lossless transfer and strict frame order. TCP is, however, sometimes considered to have lower performance due to high processing overhead and requirement to retransmit any lost packets. UDP does not have these issues, but it would require building the reliability features on the application layer. It would also be possible to implement a custom protocol from scratch using raw IP frames, but no advantage is seen in that approach, as the existing protocol stacks must already be highly optimized.

Before reviewing other protocols, we need to take a deeper look into TCP technology. In TCP the sender must store all sent packets until receiver has acknowledged successful reception. The size of this storage buffer, also known as transmission window, controls the amount of data travelling between the endpoints at any moment of time. Large window is mainly required to provide decent data rate in high latency networks, but it also allows higher traffic bursts in low latency environment. For optimal performance in different networks, window size is thus auto-tuned during communication. When packet loss is detected, TCP also drops its data rate, as loss is usually effect of reaching media bandwidth limit. The transmission speed is then again increased gradually based on an algorithm. If the loss is instead caused by a random error, data rate may remain low in vain. As these issues are most harmful when trying to achieve maximum data rate in high bandwidth, high latency networks, effect on fieldbus communication in good quality network is small. We also do not see TCP stack performance as a big issue, as virtual fieldbus would usually run on devices already utilizing TCP for other purposes. A worse problem in our point of view is that TCP was not essentially designed for real time applications and does not guarantee timeliness of transmission [68]. In a fieldbus application, dropping a frame might be better than delaying other traffic too much. One of the real-time issues in TCP is that IP frame transmission may be delayed for collecting more data in order to avoid unnecessarily large number of small frames.

Alternatives for TCP exist, but they are not very frequently adopted. To compete TCP in our use case, a protocol should achieve better real-time features while keeping the frame order and as much reliability as possible. Portability is a possible issue in using third party special protocols, if the virtual fieldbus should support also limited systems. RTP (Real-Time Transfer Protocol) resembles UDP, as it lacks the frame loss recovery mechanism of TCP [68]. Its features seem to be specific to multicast multimedia streaming [68], and thus not useful in fieldbus applications. RUDP (Reliable UDP) has taken a lightweight approach, adding only the reliable in-order delivery to traditional UDP [69, p. 3] in order to avoid many TCP issues. This variation also supports setting retransmission deadlines, but unfortunately only values greater than 100ms are accepted [69, p. 7]. SCTP (Stream Control Transmission Protocol) covers and extends the main TCP features. It is not essentially real-time targeted protocol but it has multiple useful features as retransmission deadline, framing on application layer and multi-streaming. Many studies compare its performance with TCP, often finding SCTP

throughput lower [70; 71] and latency equal or higher than TCP [72]. Advantages are seen mainly in high loss networks [73; 74], which are not the primary target environment for a virtual fieldbus. Some interesting protocols are unfortunately still at draft stage, low latency oriented UDP-RT [75] being an example. As none of the above reviewed protocols provided superior properties compared to TCP, we see no realistic alternative to using TCP for fieldbus networking transport protocol.

3.1.2 Obtaining maximum TCP performance

TCP being selected as the transport protocol, we should concentrate on reducing the effect of its disadvantages. It was already noted, that TCP is stream oriented protocol and internally makes the decision on how much data (from 0 to 64 kb) it collects before sending it in an IP layer frame. The default method for this is called the Nagle algorithm, which transmits only full TCP frames if the receiver has not acknowledged all the previous ones. It is clear that this kind of algorithm does not work well in all cases, and thus modifications have been proposed [76]. Whereas it would be difficult to change the algorithm, completely disabling it is easy using the socket option TCP_NODELAY, causing all data to be sent immediately. This has been proven to drop the communication latency over 60%, but obviously at the expense of throughput [66, p. 6-7]. For the increased traffic, this method has been generally criticized, but accepted in real-time oriented applications such as virtual fieldbus, where the initial goal of Nagle is too contradictory. With the algorithm disabled, controlled packing of multiple fieldbus frames together is still possible on the application layer. We thus see this method worth experimenting in the virtual fieldbus implementation.

It is sometimes stated, that higher performance could be achieved using multiple TCP connections instead of one. This would be a possible implementation in the fieldbus case where multiple channels are often required. The actual benefits would still be small, as showed by deeper studies [77; 78], and obviously even smaller when working in rather low latency and almost lossless network.

A TCP phenomenon called slow start occurs when the initial size selection for transmission window in the beginning of the connection is too small compared to the connection bandwidth. Good results have been achieved in latency of bursty (web) traffic by increasing this value [79]. This could apply to fieldbuses with similar traffic, but only in higher latency networks.

3.1.3 Application protocol

Regardless of the protocol used, transport layer offers a plain data carrier frame with undefined content. In fact, some protocols as TCP are of stream type and framing must be done on application layer. Essentially, the carrier frames are not identical to the frames of emulated fieldbus. An encapsulation protocol is required for common understanding of how fieldbus frame components are located in the carrier frame. Pure fieldbus traffic encapsulation is alone not enough, but the application protocol must also

support control and configuration messages for setting up and operating the virtual fieldbus.

Standardized encapsulation protocols exist, for example, in the case of IP routing [80; 81], but they are obviously not suitable for fieldbus use. Some existing fieldbus tunneling solutions as AnaGate reveal their protocol for custom applications [82]. However, no protocol seems to be widely adopted and thus no compatibility benefits are currently seen in using an existing one. For a custom protocol, there are no significant limitations to obey. As TCP does provide a checksum and error detection mechanism, those are not needed in the application protocol. Frame size information must of course be included in the protocol, as TCP is stream oriented. It is also good policy to embed protocol version to at least those messages sent when establishing the connection to avoid problems later if modifications to the protocol are introduced.

3.2 Inter-process communication

When the communication endpoints are two processes in the same computer system, a greater selection of data exchange options exists compared to the above discussed network tunneling. These techniques must be evaluated in order to achieve maximum performance. Generally speaking, concurrency and multi-threading are essential requirements of numerous software applications. The needed synchronization mechanisms such as semaphores and mutexes are basic services of any modern operating system and this makes programming rather straightforward. Working inside a single process, however, has the significant advantage of free access to common memory. If multiple processes are involved, this is no more possible due to isolated memory regions (user spaces), assigned by the operating system. Technically the most obvious option is to bypass this limitation creating a separate shared memory area, but also other methods exist.

The suitability of an inter-process communication (IPC) method in the fieldbus case relies primarily on the type of data it exchanges. Fieldbus communication is essentially a stream of rather small-sized message frames. This rules out methods for transferring or saving big data blocks with low frequency (regular files), and in the other extreme the signaling methods with no data payload, for example the UNIX signals. The realistic technologies found are listed in table 3.

***Table 3**: Inter-process communication methods in Windows and Linux platforms [83; 84]*

Technology	Availab-le in	Full-duplex	Transfer element	Notes
Pipes	Both	No	Stream	Parent-child processes only
FIFOs / Named pipes	Both	No	Stream	Multiple writers supported
UNIX message queues	Linux	Yes	Datagram	Message types supported
UNIX domain sockets	Linux	Yes	Stream	
Custom shared memory	Both	User defined	User defined	
Localhost network sock-ets	Both	User defined	Stream or datagram	Network supported
(D)COM / RPC	Windows	User defined	User defined	Network supported
Mailslots	Windows	No	Datagram	Network supported

In these technologies, differences exist in usage logic and portability. Our primary interest is still the performance they provide, in the means of data rate and latency. Figure 4 presents some published measurement results for different Unix IPC methods. Unluckily, similar results for Windows were not available, but on the other hand there is no reason to assume very different behavior when considering those methods available on both platforms.

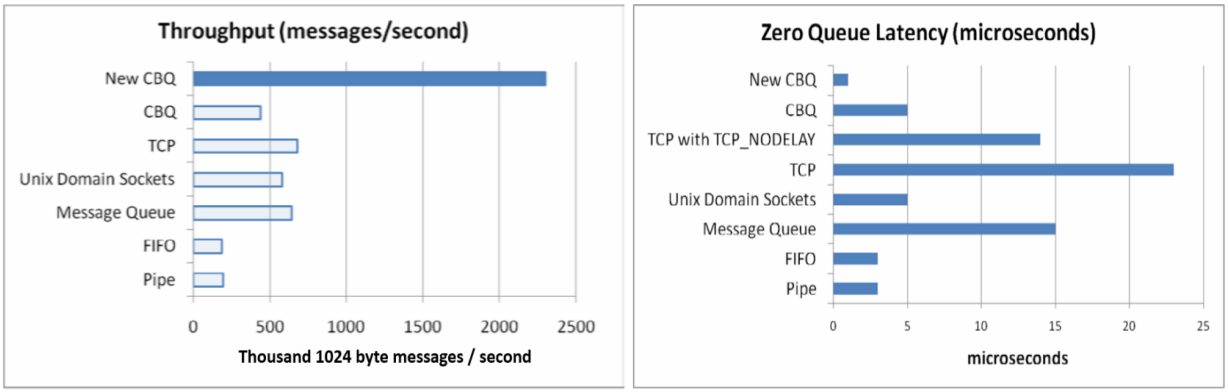

***Figure 4:** A performance comparison of UNIX inter-process communication methods by Nambiar et al. [66]. "CBQ" represents a custom shared memory communication queue.*

Two main conclusions can be drawn from the results in figure 4: First, building a custom shared memory based communication mechanism grants without a doubt the best performance for both latency and throughput. It should be the primary implementation option if most communication participants are located in the same system. Second, localhost network communication (TCP) seems to compete well with the other secondary alternatives, as pipes, FIFOs and domain sockets. Thus, no advantage is seen in im-

plementing inter-process communication with those methods, if IP networking will be supported anyway. Using network-oriented methods also locally could still possibly cut down the already limited performance in remote networking, which may further justify using another IPC method in parallel for local connections.

3.3 Cross-virtualization communication

In addition to the Ethernet- and inter-process communications, there exists a third communication type for crossing virtualization borders. As tools used for virtualization and emulation are various, we cannot discuss all of them. In the next subchapters, a deeper analysis is conducted on two alternatives, Oracle VirtualBox and QEMU.

3.3.1 QEMU

To fully understand QEMU guest to host communication, a brief introduction to QEMU technology itself is needed. Directly citing the official web page [85], "QEMU is a generic and open source machine emulator and virtualizer". It has been developed for years, the most recent version being 1.2.0, released in the year 2012. At this point we need to distinguish its two usage modes: emulation and virtualization. Emulation uses dynamic translation when running non-native binaries, and in this mode QEMU can be run in user mode without any special requirements for the host platform [86]. QEMU virtualization mode, instead, makes use of host kernel KVM (Kernel-based Virtual Machine) module or a Xen hypervisor [85] to achieve near native performance. As our use case in the upcoming implementation is to emulate PowerPC, the emulation mode will be emphasized in the following text.

Shared memory would naturally be the fastest option, probably even if it involves some mirroring or copying across between the guest and host memory pages. Also guest-guest communication might be possible directly. QEMU has an integrated shared memory device called ivshmem (also recognized with the name Nahanni), simply enabled by a command line option. It is measured to achieve throughputs as high as 2GB/s and latencies around 0.5µs, but this technology is only available in the KVM mode [48].

There exists also another early PCI-based implementation for QEMU 0.13.0, a patch called VMShm [87], but we were unfortunately not able to compile it into the later versions due to major changes in QEMU memory management. It was also tried to run our PowerPC virtualization setup on a patched 0.13.0, but the device initialization fails due to defect PCI bus of the emulated PReP machine. These experiments and gathered information show that shared memory for QEMU PowerPC emulation without KVM is not possible with unmodified QEMU. VFIO, a fully KVM independent high performance custom virtual device mechanism, could solve this problem but it is currently still under development [88].

Logically, the next communication method to examine is IP networking. QEMU offers basic virtual networking for practically all emulated machines. While any virtual or real networking solution can probably never exceed best shared memory perfor-

mance, the basic QEMU guest-host link can be significantly slow even compared to regular 100Mbps LAN (figure 6). To improve the QEMU networking performance, several possibilities exist. A paravirtualized network adapter called virtio-net [89] has been implemented to cut down the overhead caused by full virtualization. This option is possible to configure also on a PowerPC machine, and might provide a significant boost in throughput while being less effective on latency as seen in figures 5 and 6. Developers have gone even further by moving the virtio-net descriptor conversion from QEMU user space to a host kernel driver called vhost-net [90]. Using this method drops the latency to the same category with host network (figure 5), which should be enough for most network-based virtual fieldbus solutions. However, vhost-net cannot be enabled on a non-KVM guest.

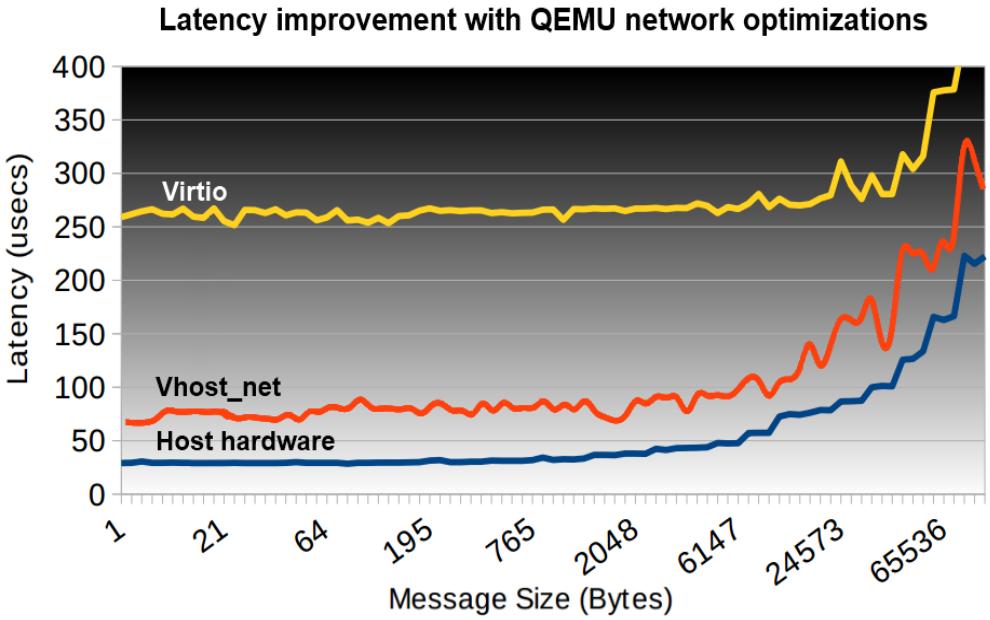

Figure 5: *QEMU network optimizations - effect on latency [88]*

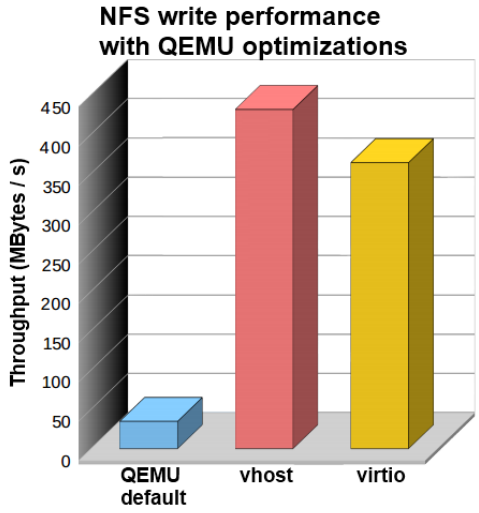

Figure 6: *QEMU network optimizations - effect on throughput [88]*

3.3.2 VirtualBox

VirtualBox is a “general-purpose full virtualizer for x86 hardware, targeted at server, desktop and embedded use” [91]. Its key differences to QEMU are higher level of productization, official support for Windows host and support for x86 platforms only. Again, we primarily seek a technique of sharing memory between host and guest to obtain maximum performance. VirtualBox supports a mechanism called Guest Additions for machine interaction. The addition drivers are installed on top of guest OS and they are used for features like shared folders and clipboard or mouse pointer integration. Regarding memory, the provided features called Memory Ballooning and Page Fusion aim at efficient physical RAM utilization. Page Fusion is actually based on shared memory, but it cannot be used as a communication method, as only static identical memory regions are shared. [49, p. 57-74]

As official method for memory sharing is not provided, it still could be custom-built, as VirtualBox is open source software [92]. Such implementation should be based on fixed interfaces to be compatible with future versions. One possibility might be the Host-Guest Communication Manager interface (HGCM), referenced in many discussions on internet forums, but no official documentation for it was found. A thorough article by Kurakin goes even deeper, sketching a technique of sharing memory based only on the assumption that guest memory is physically part of host memory [93]. To outline, using shared memory between VirtualBox guest and host is without a doubt laborious, even more difficult than with QEMU.

Data types supported by other official sharing mechanisms like clipboard and file sharing are not suitable for fieldbus traffic. Again, the well-supported virtual network between machines seems to be the easiest option, if performance is found feasible.

3.3.3 Preliminary cross-virtualization measurements

Regarding both virtualization tool alternatives, network was detected the best supported option for host-guest communication. Implementing any other communication method seems very troublesome and cannot thus be included to the upcoming implementation just for measurement purposes. For this, basic measurements on network performance were conducted before final design decisions. Unfortunately we could not get virtio-net and time counters working in the same QEMU setup, so the effect of optimization could not be measured.

Throughput was measured using the popular free test suite iperf [54], which has support for both Linux and Windows. The Linux version was cross-compiled for a PowerPC target to be usable in the QEMU environment. For latency measurement, an implementation of traditional ping tool is provided on most platforms. On Linux, its resolution is in microsecond level, but the Windows default tool only has millisecond accuracy. For this reason the free utility hrPING [94] with better accuracy was instead used on the Windows host. The machine used for the measurements is specified further

in section 5.1. Results are presented in figures 7 and 8, where the horizontal axis positive extremes are scaled up to fit the low and high results in the same figure.

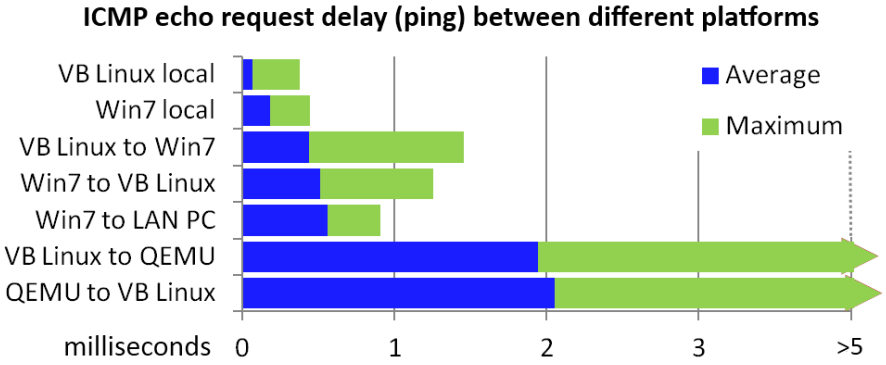

Figure 7: *Preliminary latency results in virtualized networks*

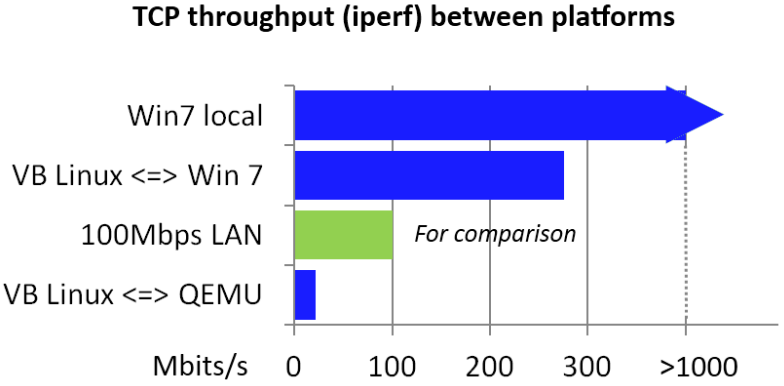

Figure 8*: Preliminary throughput results in virtualized networks*

From the results we see that VirtualBox has very good network performance, and implementing a custom build with cross-virtualization shared memory support would thus be an extreme solution. Regarding QEMU, the performance is much lower but still found adequate for fieldbuses with lower requirements. The measurements were carried out in a worst case scenario with nested virtualization and no QEMU network optimizations, so we also see plenty of room for improvement.

3.4 Portability

Full featured virtual fieldbus can naturally be built with only single operating system supported. It is also possible to implement separate versions of the bus components for different platforms, sharing a common protocol for intercommunication. However, implementing the abstract part and maintaining the project becomes a lot easier if common code can be used on all platforms. Multiple third-party libraries are provided for easy portability, for example Boost and GLib [95], but some of them are not free for commercial use, excessively wide or too heavy for embedded devices. As we are building a

library too, the dependencies to secondary libraries might result in linking difficulties in the user application. Use of dynamic memory allocation in the libraries is disallowed according to MISRA-C, often used in embedded programming [96]. Thus, implementing custom lightweight portability wrapper for the platform-specific required concepts seems to be the most flexible solution.

In the practical portability discussion in the next chapters, we will use C/C++ examples, as those languages clearly dominate in the embedded programming [97].We will also restrict the discussed platforms to Linux and Microsoft Windows, as they are seen the main options across virtual fieldbus target devices.

3.4.1 Concurrency and IPC

As found when evaluating possible IPC mechanisms, such services provided by operating system vary a lot on different platforms. For communication, shared memory was detected to give the best performance, but portability was not yet evaluated. To support the IPC communication and to allow multiple ongoing transmissions on different medias, also synchronization and multi-tasking is required. For concurrency programming both Windows and Linux introduce a wide set of mechanisms, as mutexes, semaphores, events and threads. In Linux, we have the additional option of choosing between System V and POSIX technologies, both providing roughly the same mechanisms. As POSIX is fully multi-thread safe [98] and has simpler interfaces, we will not consider System V in the following text.

The interfaces on Windows and POSIX are of course different, but more challenging diversity is found in the concepts themselves. This prevents creating a portability wrapper for some of the mechanisms, as common concept does not exist. Table 4 lists the reviewed mechanisms and the key differences [95; 100].

***Table 4**: Comparison of system service concepts on Wnindows and Linux*

Mechanism	**Linux/POSIX implementation**	**Windows implementation**
Named shared memory	**Functions:** *shm_open, mmap, shm_unlink* **Notes:** Integer descriptor, errors reported via errno	**Functions:** *CreateFileMapping, MapViewOfFile, UnmapViewOfFile* **Notes:** Handle descriptor, errors reported via GetLastError()
Local unnamed mutex	**Functions:** *pthread_mutex_init, pthread_mutex_lock, pthread_mutex_unlock, pthread_mutex_destroy* **Notes:** Initialization can be substituted with assignment. Special attribute needed for deadlock prevention.	**Functions:** *InitializeCriticalSection, EnterCriticalSection, LeaveCriticalSection, DeleteCriticalSection* **Notes:** Deadlock caused by consecutive lock calls prevented by default.

Shared unnamed mutex/ semaphore	**Notes:** Any unnamed object can be shared in memory.	**Notes:** Sharing unnamed object between processes seems not to be possible.
Shared named mutex	**Functions:** - **Notes:** Not available as such. Use shared named semaphore instead.	**Functions:** *CreateMutex, WaitForSingleObject, ReleaseMutex, CloseHandle* **Notes:** The Windows mutex resembles Linux named shared semaphore with maximum count restricted to 1.
Shared named semaphore	**Functions:** *sem_open, sem_wait, sem_post, sem_close* **Notes:** Post function increases semaphore count always by one.	**Functions:** *CreateSemaphore, WaitForSingleObject, ReleaseSemaphore, CloseHandle* **Notes:** Release function increases semaphore count by given amount.
Thread	**Functions:** *pthread_create, pthread_cancel* **Notes:** Task function prototype returns void pointer. Waiting task termination requires a special condition variable.	**Functions:** *CreateThread, TerminateThread* **Notes:** Task function prototype returns DWORD. Waiting task termination supported by default.
Sleep	**Functions:** *usleep* **Notes:** Microsecond resolution	**Functions:** *Sleep* **Notes:** Millisecond resolution

The summary clearly shows the main difficulties involved in shared memory communication and concurrency portability. Regarding process synchronization objects, using a named concept is the only option, as Windows does not support sharing unnamed objects. Creating a portability wrapper for other concepts is possible, but some tradeoffs must be made, for example, in sleep resolution and semaphore features.

3.4.2 Networking

For networking, the portability of programming mechanisms is way better. This originates from the strong concept of sockets, originally designed in 1983 in Berkeley. Independently written implementations of the Berkeley socket API services are provided for Linux with full compatibility [101] and for Windows with some modifications. The frequency of the portability issue on Winsock API has led to multiple excellent articles and tutorials about the subject [102; 103; 104; 105]. In table 5 we summarize the differences that are seen most significant among the publications.

***Table 5**: Comparison of socket interfaces on Windows and Linux*

Subject of difference	Linux implementation	Windows implementation
Error passing	Variable named errno	Function named WSAGetLastError()
Initialization and cleaning	None	WSAStartup(), WSACleanup()
Functions with non-BSD-compatible name	None, names for reference: fcntl/ioctl poll close	Multiple: ioctlsocket,WSAIoctl WSAPoll closesocket
Include differences	Typical: sys/types.h, sys/socket.h, netinet/in.h, arpa/inet.h, netdb.h	Selectable: winsock.h (winsock2.h) (Ws2tcpip.h)
Socket datatype	Integer, invalid when -1	Void pointer, invalid when equals INVALID_SOCKET
Behavior when connection is terminated while receiving or sending	recv() returns 0 send() returns -1	recv() returns 0 or -1 send() returns 0 or -1

The summary shows, that implementing a BSD compatibility wrapper for Winsock is not possible, mostly due to startup, cleanup and error functions. However, a compatibility wrapper to be used on both Linux and windows is very trivial to create.

3.4.3 Interfaces and libraries

Use of libraries is a method of portability in itself as it separates the compilation of the virtual bus code from the compilation of the user application. Dynamic linking of the library has many additional benefits, being shared between applications and loaded only when required. On the other hand, using many shared libraries with different versions can decrease installation compatibility. For this reason it is good to support both dynamic and static linking. From the virtual bus interfaces, widest portability is required for the client interface, which is used also on the embedded virtual or hardware devices with restricted Linux environment. This is luckily not hard to achieve, as Linux is heavily library-oriented system and the same mechanisms are available also in the most limited versions.

A disadvantage of using dynamically loadable libraries is the lack of proper support for C++ objects. Class methods cannot be brought directly accessible for the library user application, which prevents regular object construction even when class declaration is in the public library header. A rather simple workaround for this is to provide a plain factory function for class instantiation [106].

Program code and compilation is only slightly different on Windows and Linux platforms, at least when using C/C++ language. Windows platform requires an addi-

tional directive dllexport/dllimport to all interface functions of the library [107]. The GCC compiler family can be used on both platforms with minor differences in used directives [108; 109]. Related to that, biggest differences between the two platforms are seen in the naming convention and versioning. A good way to avoid complicated build is to use CMake, “the cross-platform, open-source build system” [110].

4 A VIRTUAL CAN BUS SOLUTION

Besides the scientific purposes, the goal of the preceding theory and technology study has been to enable a real-world implementation of a virtual fieldbus. This section describes the design with its background and introduces the software components involved.

4.1 Background

Wärtsilä, the orderer of this thesis, is well-known for large diesel and gas operated internal combustion engines (figure 9). Their solutions are targeted both to marine industry and power plants, sharing the requirement for high efficiency and environmental sustainability. Some of the recent milestones have been the world's first LNG (liquefied natural gas) powered passenger vessel Viking Grace [111] and world's largest engine-driven power plant in Jordan [112], both running on Wärtsilä multi-fuel engines. In addition to the physical products, Wärtsilä also provides full support services for the complete lifecycle of their engines.

Figure 9: *Different models of Wärtsilä 64 medium speed engine produce power of 12900..17200kW and weigh 233..295 tonnes [113]*

For the engines and their installation environments, a lot of automation technology is required. During the last decade, the mechanical and hydraulic control components used in traditional engines have been substituted with electronics and software. This has allowed better performance by smart control and monitoring. UNIC (Unified Controls) is Wärtsilä's solution for robust and reliable embedded control. The UNIC control modules form a distributed platform, which provides a powerful set of services to be used by control applications. Modularity and scalability have been its design principles to allow versatile applications concerning both engines and other automation systems. The system is illustrated in figure 10 [114, p. 40-41].

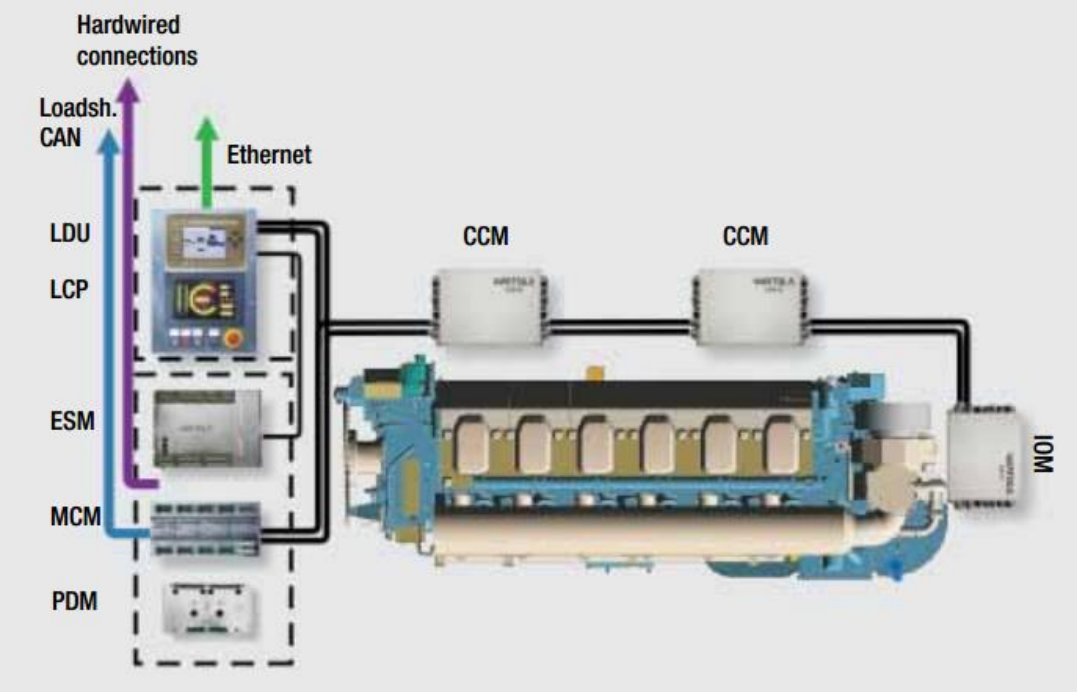

Figure 10: *The principal structure of Wärtsilä UNIC automation system [115, p.13]*

These systems are heavily dependent on fieldbuses of multiple types. CAN is one of the widest used communication method, and possibilities of virtualization have thus been of interest for long. The virtual bus implemented and studied during this thesis will be used in multiple development projects, both in manual and automatic testing.

Wapice Oy is a fast-growing technology company, providing services both in software engineering, electronics design and consultation for its industrial customers [116]. As the primary employer for this thesis, Wapice offers its facilities and long-term expertise in fieldbus technology and research [117] to support the process. The published results of the study are interesting also for them considering related future projects.

4.2 Overall design

Requirements for our implementation widely cover the features discussed during the theoretical and technical parts. The target environment is however restricted to small-scale fast networks, which rules out some technologies only applicable with slower or uncertain connections.

System topology is of course restricted by the need of a central element, typical to CAN. In our design, it is called a *hub*. Although this means compromising the best performance between two clients, it is also seen as a good thing for practical connectivity. First, the clients only have to know the location of the hub in order to connect, and second, the hub can be freely set up in the network location with best accessibility considering firewalls.

In the bus configuration and control, it was decided to follow the roles different components have in a physical hardware implementation. A real-world fieldbus device does not have any control over the physical connections. Thus, also a virtual fieldbus client can only reveal one or multiple fieldbus ports and only control their traffic by accepting or discarding frame transmission and reception. The transportation of frames between the ports of the clients according to bus layout is then responsibility of the hub, analogous to physical cabling. When the bus layout needs to be altered or defects simulated, it is logically done using interfaces provided by the hub.

4.3 System components

A closer look to system structure will be taken in the next chapters. In addition to actual software components, we discuss some of the main interaction processes.

4.3.1 Client library

A client library is the component that primarily allows a client application or a virtualized device to join the virtual bus. For some of the applications it is still only one option to use virtual bus, and thus direct connectivity to CAN bus adapters without a virtual bus installation is required. This was the main reason to create the virtual CAN support as a part of existing Wapice library, LowLevelCAN API. The library was originally implemented as an earlier thesis work to build a CAN diagnostics tool application [117], and has support for multiple hardware devices including Kvaser, IXXAT and SocketCAN. The position of the library in the virtual CAN concept is pictured in figure 11.

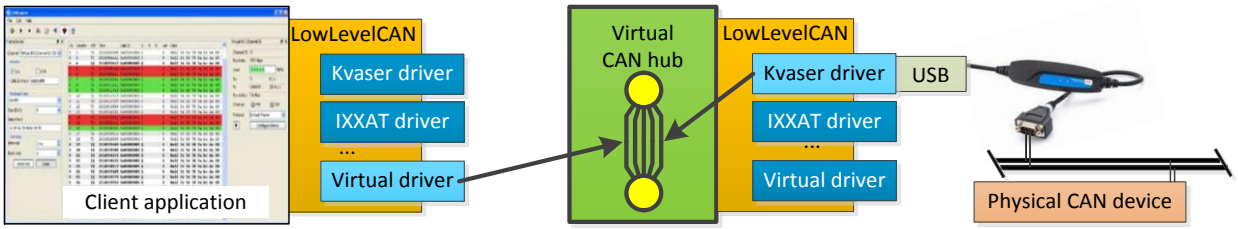

Figure 11: *Utilization of the client library in our virtual CAN solution*

In addition to the software clients, also hardware buses need to be connectable on the virtual bus. Logically, we use the above mentioned library also for this purpose. It would have been possible to create a special software client to bridge the two drivers together, but for better performance it was decided to integrate LowLevelCAN also directly to the virtual CAN hub.

4.3.2 Hub

As stated above, the virtual CAN hub acts as the central element to which every client connect when starting to use the virtual bus. It has three responsibilities: to accept, listen and serve clients of various types, to implement the multi-channel traffic logic and to provide an interface for bus configuration and control. From the hub point of view, there are three different types of clients: shared memory, TCP and hardware. In spite of the very different technologies, they all share the common interface of sending and receiving data. Thus, they are implemented using inherited C++ objects, hiding the communication mode completely from the other parts of the hub. The principle of the hub is presented in figure 12.

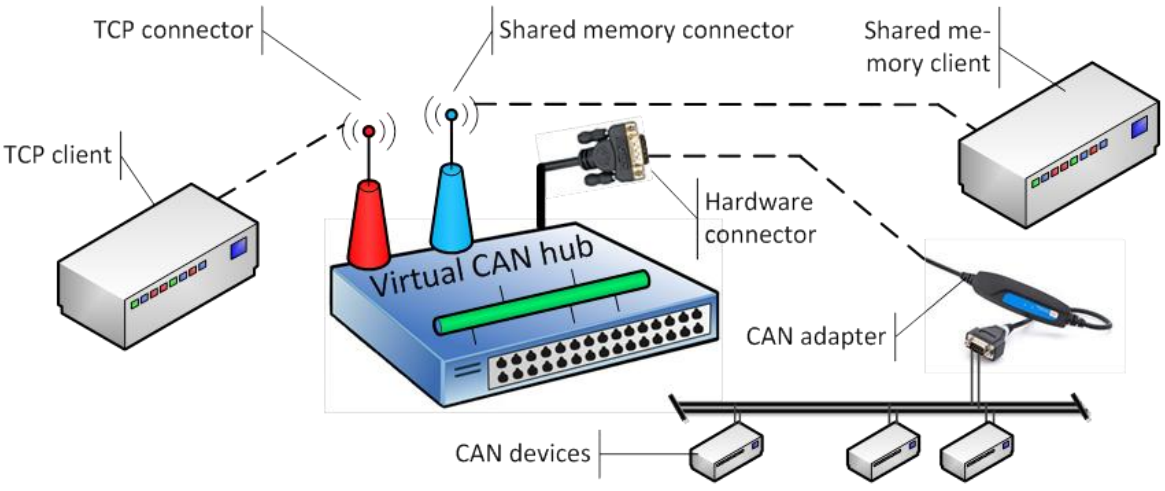

Figure 12: *Virtual CAN system components*

4.3.3 Traffic control

As defined above, traffic control is an important responsibility of the hub. The hub was designed to handle multiple channels, multiple ports on each client and even multiple fieldbus types inside the same virtual bus for future applications. To provide control over specific client's traffic, an optional client identifier is passed to hub when client registers itself on the bus. This data in total enables setting up any bus configuration by using routing rules. No complexity is added even by the requirement of runtime bus control, as it practically only needs an interface to add or remove routing rules on the fly. These interfaces and a graphical UI to operate them are however not implemented in the scope of this work. Figure 13 presents the principle of routing in our virtual CAN system.

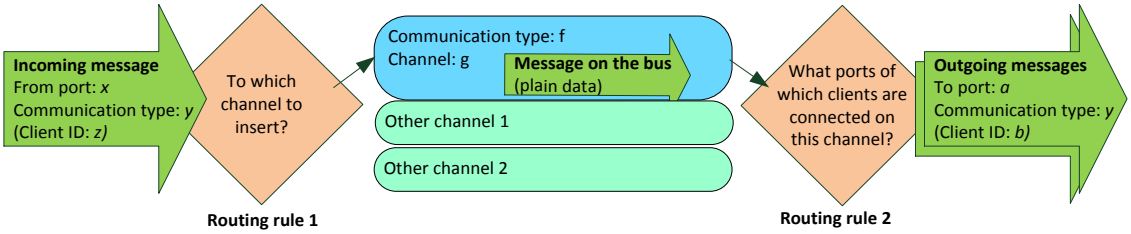

***Figure 13**: Traffic routing in the virtual CAN system*

4.3.4 Data flow

One of the main design problems in our virtual CAN system was to enable fluent data flow from client to another with as low latency as possible, still without excessive consumption of computing power. As the essential communication methods are decided based on the pre-study, this design concerns mostly the threading and synchronization. Proper understanding of these mechanisms is also essential in order to assess the performance results reliably later.

From the client point of view, TCP communication is straightforward. Shared memory is more interesting, as this method is fully tailored to this solution. We used two lockless shared ring buffers between client and hub to allow bidirectional traffic. Informing the hub about new frames from any of the clients was implemented using a single shared semaphore. For informing the clients, a semaphore is required for each client.

An example flow for a shared memory client is presented in figure 14. As each client type requires waiting of a different object type (socket, semaphore and callback), a dedicated listener thread is established in the hub for all of them. Both the TCP and shared memory clients are however designed so that a single thread can wait for multiple clients. When frames are received by the listener tasks, they pass the data to a multi-producer ring buffer and increase the bus semaphore to inform about update. Eventually, the main bus task is processing each received frame and passing it to the send methods of selected clients according to routing rules.

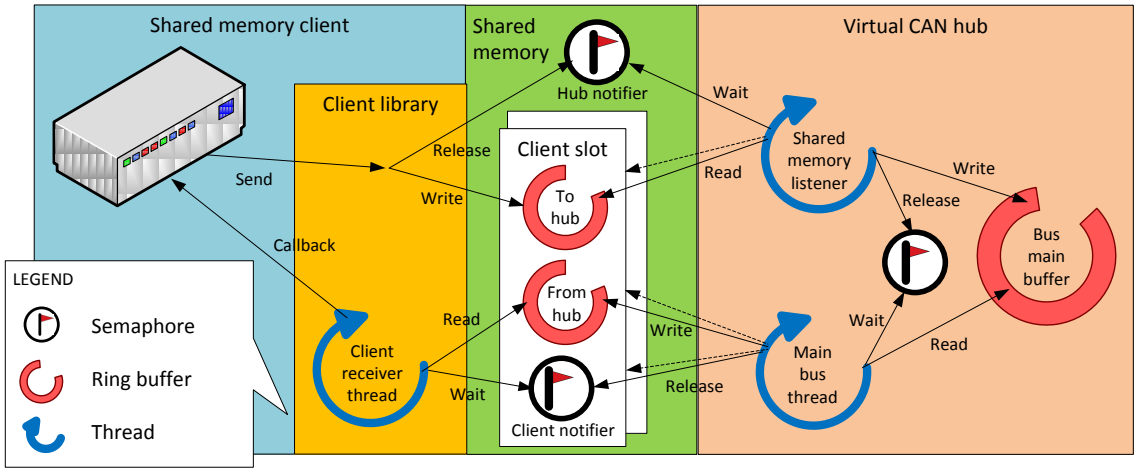

Figure 14: *Data flow in virtual CAN system with a shared memory client*

4.4 Experiences

The use of an existing CAN library was proven to greatly accelerate the virtual bus development. The only new features required there in addition to the virtual driver itself, were for querying available channels from virtual CAN hub. As no complicated data structures were involved, implementing them was mainly straightforward. Only lockless ring buffers in the hub and shared memory needed some special attention. The effort required for creating the portability wrappers for platform-specific operations was unexpectedly high, although many compatibility issues were detected already in design phase. From the different client communication forms, TCP was the most laborious due to the protocol parsing and generation involved. Also correct network connection termination and error handling required significant amount of code to work properly.

5 TEST SETUP AND MEASUREMENTS

This section explains the plan for testing and evaluating the virtual CAN implementation. The experiments of course also validate part of the functionality, but we will focus on the performance measurements.

5.1 Test environment

In order to get realistic and useful results, the software and hardware environment for measurements must closely resemble the environment where the solution is to be used in. Thus, we select an ordinary developer laptop workstation as the platform for the tests. Although some parts of the solution are rather targeted to high performance server environment, they are also measured on the same platform to allow better comparison. Detailed machine and software specifications are found in table 5.

***Table 5**: Test environment specifications*

Machine specification	
Model	HP EliteBook 8540w
CPU	Intel Core i5 560M @ 2.67 GHz
Memory	8GB DDR3 @ 533MHz
Chipset	Mobile Intel QM57 Express Chipset
Graphics	NVIDIA Quadro FX 880M, 1GB
Software specification	
Windows version	Windows 7 Professional 64bit SP1
Linux distribution	Ubuntu Linux (12.04)
QEMU version	0.14.0

Different virtualization levels are the most important subject for measurements due to the possibly decreased performance involved. Thus, a nested virtualization platform consisting of QEMU on top of VirtualBox was built. The hubs and clients were then set up in all possible locations with all communication methods to achieve maximum coverage of measurement combinations. The complete variety of the locations is presented in figure 15, where an identifier tag (in quotes) is also assigned for each location for later reference.

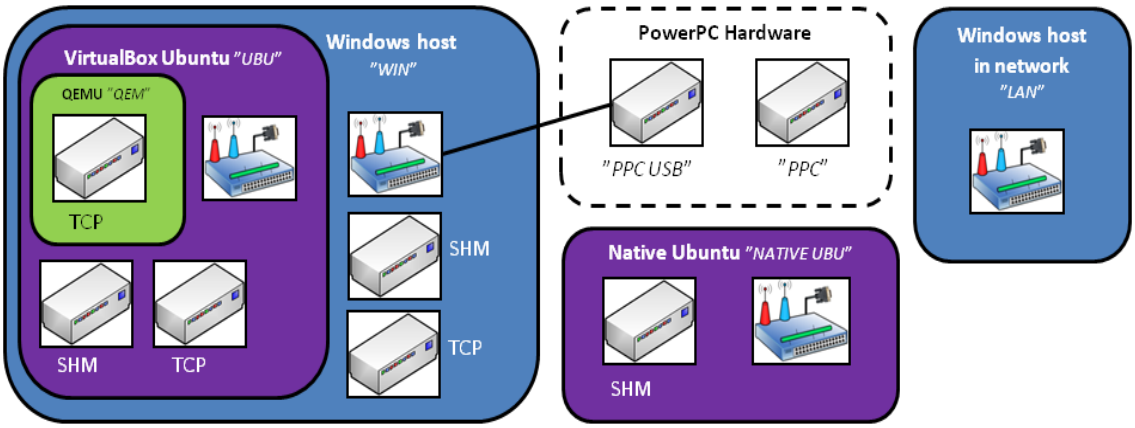

Figure 15: *Communication participants in the measurements*

To measure the performance in a local area network, a simple point to point 100Mbit/s Ethernet link was set up between two PCs. This was seen as a good method to stabilize the environment, as only few components and no other traffic were involved. For hardware bus interconnection measurements, a PowerPC device was connected on a 500kbps CAN bus, which was then connected to the PC workstation using a Kvaser Leaf Light USB CAN adapter.

As no very clear results on practical CAN bus hardware latency were found in the theory study, we also measured the latency between two hardware clients with no virtual bus involved. This allows better comparison with other results. Also another comparative measurement target was added based on the initial results: a native Linux installation was created for improved shared memory result coverage.

5.2 Measurement methods and variations

As our implementation uses custom protocol and interfaces, third party benchmark tools cannot be utilized. For this reason, simple measurement programs were constructed for each purpose. The following chapters specify the custom methods and values used in the measurements.

5.2.1 Latency

As discussed earlier, latency measurement is most useful if conducted in a statistical manner and presenting a distribution instead of plain average value. To achieve these kinds of results, great number of accurately measured latency samples is needed. Clocks are not necessarily synchronized in the virtual environment when using separate devices and we thus will measure round-trip times instead of one-way delays. Any other bus traffic was removed during the tests.

A small test application was programmed for this purpose. It integrates with the client library, searches for available channels and connects on the bus. The latency measurement procedure is presented in figure 16. For the sample count we selected 100 000 to guarantee comprehensive statistics in reasonable time. During the test, results are stored in memory and written to a text file when process is completed.

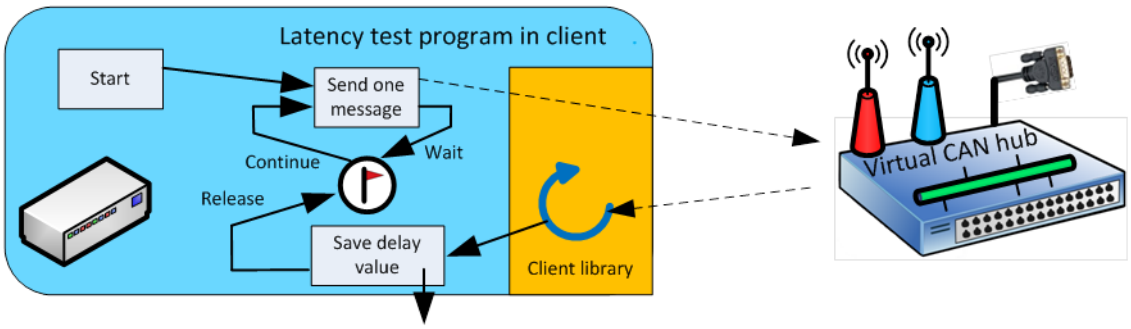

Figure 16: *Latency measurement process*

Time measurement with microsecond resolution was detected as a portability issue. Time source must also be a linear counter to rule out any external time changes. On Linux platform, these requirements are relatively simple to meet using the function clock_gettime [MAN]. Windows does not provide a straightforward method, but we were able to create a port of clock_gettime using QueryPerformanceCounter and related functions [MSDN].

The virtual CAN hub is initially suitable for this measurement technique. Instead, the comparative measurement using hardware devices required another implementation for the communication partner which echoes the measurement frames back to the bus. The similar test application was attached directly to the CAN driver of the devices to reach minimum latencies. Naturally, the client library or any other virtual bus components were not used in these measurements.

5.2.2 Data rate

Considering data rate, only the maximum capacity is measured as traffic limitation mechanisms were not yet implemented. The measurement setup is similar to the latency measurement with only one hub and client involved. The client side test program is again illustrated graphically in figure 17.

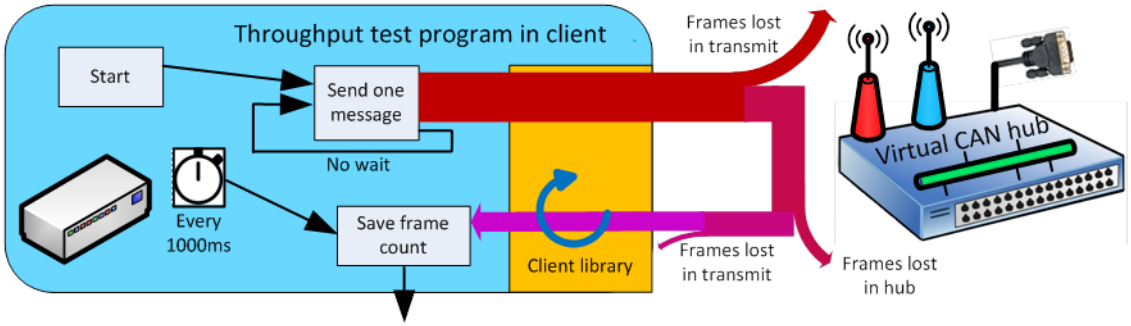

Figure 17: *Throughput measurement process*

In this case time measurement accuracy is not as big issue as with latency, but we however use the above described methods. Although the results are in the end combined

into an average value from the complete timeframe, the test program saves the result separately for each second. This way it is possible to validate the process stability afterwards. 60 seconds was selected as the duration of the entire test.

5.2.3 CPU load

Due to high effort required for the important latency and throughput tests, less attention was left for the CPU load measurements. The scope was thus restricted to two platforms, the host Windows and Ubuntu Linux in VirtualBox. The purpose of the measurements is to find out the load difference between shared memory and TCP communication, and to evaluate the effect of virtualization.

This time, the test program was required to simulate certain level of bus traffic in order to get comparable results. The simulation was implemented by adding an adjustable delay in the message sending loop. A delay value was experimentally selected to create 500 kbps traffic. The test was conducted in all cases using both one and three clients. Also in the latter case, traffic was created by one client only. The process is illustrated in figure 18.

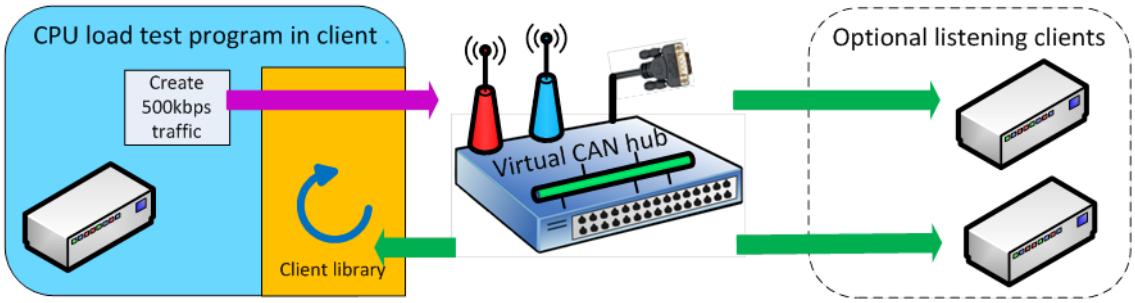

***Figure 18**: CPU load measurement process*

The CPU load itself was observed using the default tools of both platforms: *Process Manager* on Windows and *top* on Linux. The results were scaled so that 100% equals full load on all cores.

6 RESULTS ON BUS PERFORMANCE

The experiments carried out according to the previous section resulted in high amount of useful data. In this section these results are presented and explained using illustrative graphics. The important findings are pointed out, but further discussion will be performed later in the next section.

6.1 Latency

The 100 000 samples recorded in each measurement were sorted in Microsoft Excel to observe the probability of the latency to fall below certain value. In the sorted list, this probability can be read directly from the row number for the latency value of each row. For visualizing the results, we selected the values associated with probabilities of 80%, 95% and 99% rather than presenting the complete distribution. The results are presented in figure 19.

Results are grouped according to the location of the hub, and sorted from low to high latency inside the groups. Comparative results on pure hardware and native Ubuntu installation are presented last. The horizontal axis is divided into two sections to allow better visibility for lower results, still fitting the high USB adapter result into the same figure.

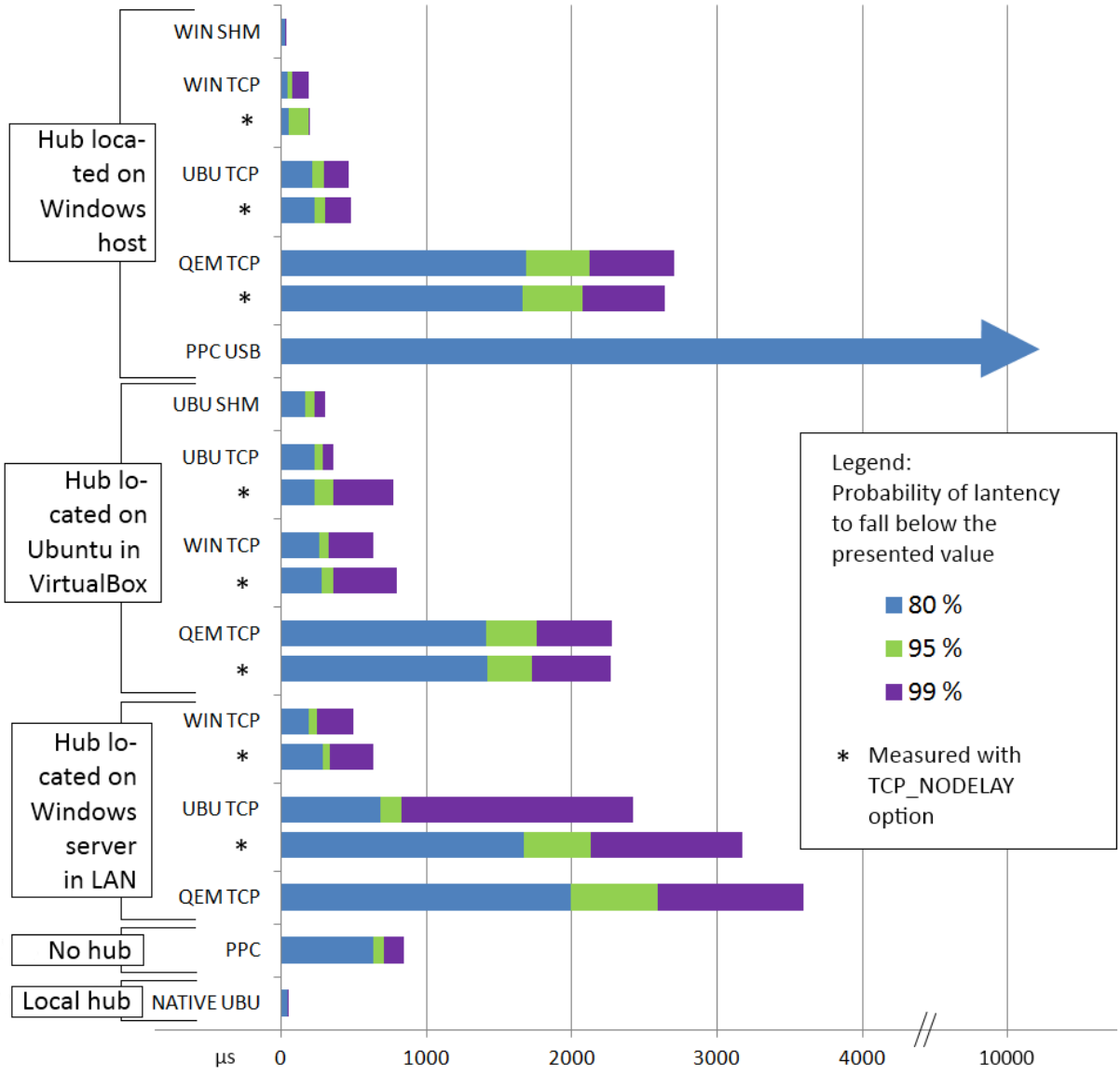

Figure 19: Measured latency results on our virtual CAN solution

Observing the latency results, first thing to notice is the vastly higher result on USB adapter based connection. This is something unexpected, as no virtualization was involved in the experiment. The over ten millisecond round trip time is a potential problem when interconnecting virtual and hardware fieldbuses.

In the search of optimal TCP connection latency, the use of TCP_NODELAY option was experimented. However, the results prove only minimal improvement when the client is located in QEMU whereas it caused slightly decreased performance on Windows client and significantly poorer results on the virtualized Linux client.

When comparing other results to the hardware performance presented, it needs to be noted that hardware delay was measured directly between clients and it thus represents twice better performance than the same value in virtual CAN latency. In any case, we find all single-PC results on VirtualBox and native systems in the same class with the hardware latency. Even local area network between native hosts is well usable for

bus virtualization. Shared memory is found substantially faster than a hardware bus and will thus suit even to extreme applications.

Virtualization still has a clear increasing effect on the latency despite of the communication method used. VirtualBox decreases the shared memory performance dramatically while the comparative result on native Linux proves that our implementation makes no significant difference in the two operating systems. Using QEMU, at least using it inside VirtualBox, causes over two millisecond latencies, which may limit the successful use cases.

6.2 Data rate

Throughput results are slightly simpler than above presented latency values, as only one average is associated with each result. Throughput of the hardware bus is known by definition and thus not measured or presented. The results are presented in figure 20, again grouped according to hub location and with divided horizontal axis to fit the small and large results into same figure.

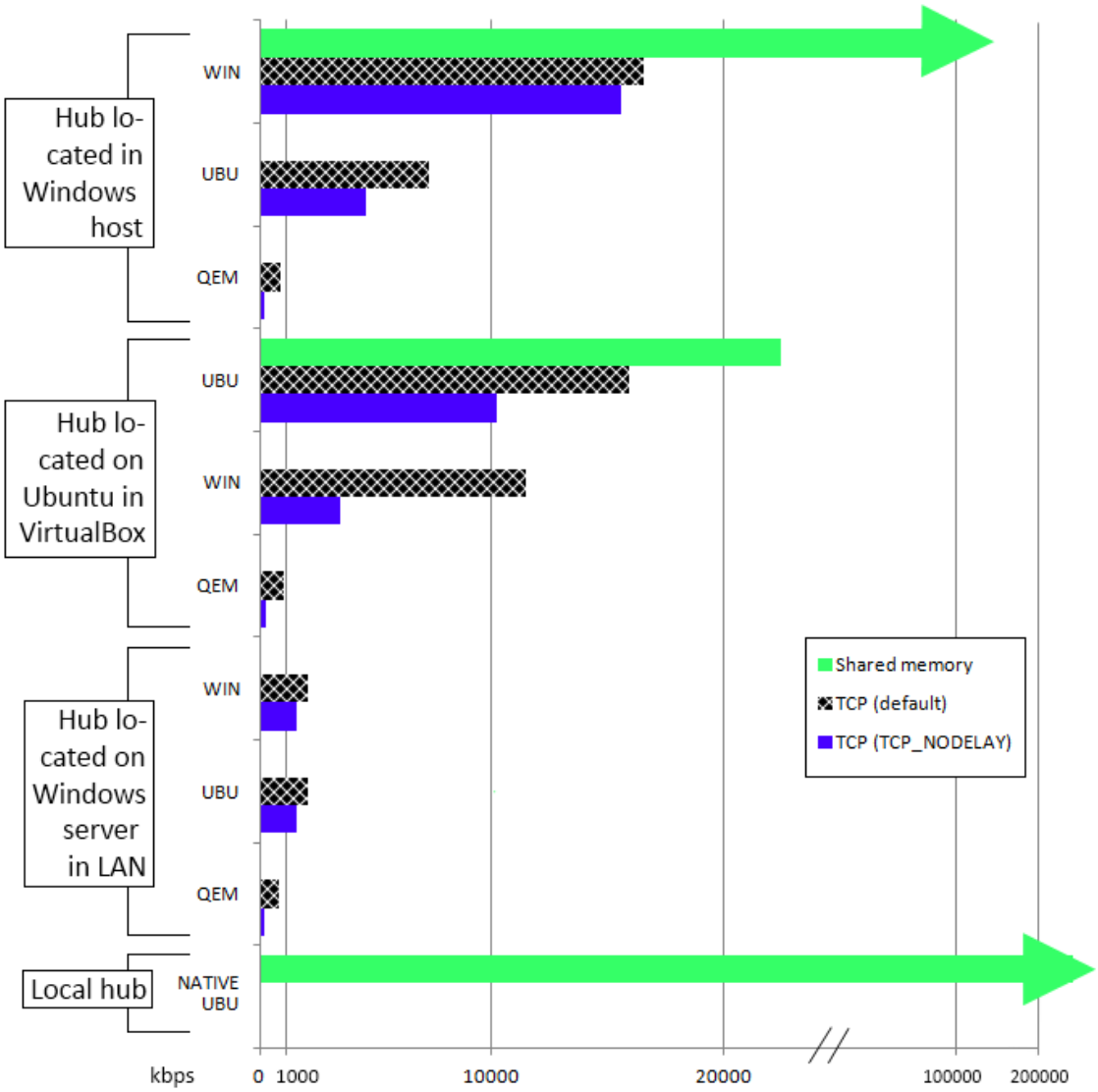

Figure 20: *Measured throughput results on our virtual CAN solution*

Generally said, the data rate results agree well with the ones of latency. All the single-PC methods excluding QEMU provide enough bandwidth for multi channel operation, even when 1000kbps CAN speed is used. Also here, virtualization cuts the shared memory performance radically, but has smaller effect on TCP communication. Surprisingly, shared memory throughput was almost doubled on native Ubuntu platform compared to Windows.

TCP_NODELAY option was questioned already based on the latency results, and this finding seems yet clearer when observing the dramatic drop it causes to data rate. Using TCP_NODELAY makes QEMU completely unusable for the fieldbus purposes, as the bandwidth does not suffice even for single 500kbps CAN line.

The results measured in local area network show that our implementation over TCP is far from perfect, when the network itself easily provides speeds in the class of ten megabits per second. This is quite obvious, as we did not use any packing of frames in the measurements but each frame was sent alone. The same requirement for any possi-

ble optimization is valid also for QEMU, which currently does not suit for multi channel CAN operation on full load.

6.3 CPU load

The CPU utilization results have slightly increased possibility for error, as there was some inconsistency between the values reported by Linux in VirtualBox and the host Windows running the virtualization. Results are presented in figure 21, categorized by the platform and whether shared memory (SHM) or TCP is used. The alternative results with different count of listening clients are represented by overlaid bars.

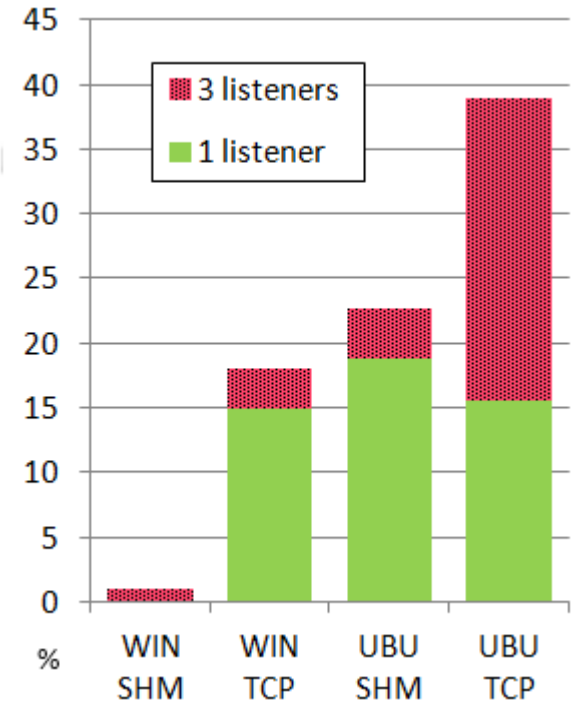

***Figure 21**: Measured CPU load results on our virtual CAN solution*

CPU results verify the virtualization influence to shared memory the third time. Whereas the shared memory communication on native Windows causes only negligible load, the resource requirement for same traffic multiplies in VirtualBox. Similar effect is visible but not as dramatic with TCP. On both platforms, the results strongly rationalize the use of shared memory, even in parallel with TCP communication required for remote connections.

7 CONCLUSION

We have now gone through a complete research process on virtual fieldbuses, from theory to application and evaluation. Every chapter has brought valuable information for achieving our goals and now it is time to form the end conclusions. This chapter will provide clear answers to the research questions stated in the beginning and also evaluate the thesis process.

7.1 Technology selections

Technologies were reviewed and evaluated from the performance, portability and applicability aspects without forgetting the effort required for utilization. Regarding data transfer, methods were searched for communication in single OS between processes, in local area network and between virtualized machines. Based on existing publications, shared memory was assumed to provide highest speeds and the assumption was proved correct with our measurements. Shared memory should thus be the primary option for local communication technology, since it is also easily portable and causes less CPU load compared to TCP.

Network communication will obviously be based on IP to utilize existing physical and virtual networks. Different transport protocols were reviewed, but in the end it was seen that TCP will suit best in virtual fieldbus use due to its reliability, performance and availability, despite of the non-optimal real time behavior involved. Based on existing results, we assumed that the TCP_NODELAY option would optimize TCP latency. This was proved wrong in the measurements, as no significant latency improvement was gained while the option caused throughput to plummet.

For crossing virtualization borders, using virtual network and TCP was found the most compatible method. In fact, VirtualBox does not provide any good alternatives to using it, since implementing for example a shared memory region between guest and host would require a custom build of VirtualBox. QEMU instead provides a built-in shared memory device called ivshmem, which would probably be the best option. Also for network QEMU has powerful optimizations called virtio-net and vhost-net. Some of them unluckily require QEMU to be used in the KVM mode which is unavailable in nested virtualization and measuring their performance was thus not possible in this study. However, using any of these optimizations seems to be the key for proper QEMU fieldbus performance.

Significant performance bottlenecks were found in two areas of technology. Nested and non-optimized virtualization, particularly using QEMU in emulation mode inside VirtualBox in our case, quadrupled the latency and caused even greater drop on

throughput. The root cause for the issue was the inability to utilize any of the above mentioned QEMU communication optimizations. To avoid these problems, virtualization layout should be kept as simple as possible and the compatibility of QEMU optimizations should be verified before including it in design.

Unexpected and critical performance decrement was found in the use of a USB CAN adapter, which caused high latencies. This finding is presumably not true for all device models, but places a serious concern and requirement device performance verification before building a virtual bus dependent on fieldbus adapters.

7.2 Achievable performance

An important research question was to find out how close to hardware performance it is possible to get with a virtual fieldbus solution. The performance evaluation was carried out by measuring both latency distribution and maximum throughput. Virtual CAN bus was seen as a good selection for evaluation, because its hardware implementations were found to provide lower latencies compared to several other fieldbuses. However, the results and drawn conclusions apply well to any fieldbus as long as the differing timing and bandwidth requirements are taken into account.

As expected, the performance was found highly dependent on the technologies and environment used. This is summarized in table 6, where different conditions are linked to the achieved performance.

***Table 6**: Achieved performance in different conditions*

Environment characteristics	**Achieved round-trip latency**	**Achieved throughput**
Native OS installation	< 100 µs	> 100 Mbps
Linux in VirtualBox	~ 300 µs	~ 20 Mbps
Crossing VirtualBox border	~ 500 µs	~ 10 Mbps
Native OS machines in LAN	~ 500 µs	~ 1 Mbps
Nested virtualization in QEMU (without optimizations)	~ 2 ms	~ 900kbps
Slow USB fieldbus adapter	> 10 ms	(fieldbus dependent)

The measurements were conducted on our solution, where frame packing or similar optimizations were not yet implemented. Because of this, we also see room for potential improvements in the throughput results at least on network communications.

7.3 Suitable application areas

In the early discussion, potential applications for a virtual fieldbus system were seen on multiple fields. A successful implementation would allow virtualized developing and debugging distributed systems and also automatized system level testing without any

real hardware or complex physical test instrumentation. Virtual bus technology was also seen as a possible method to interconnect isolated locations via a fieldbus tunnel, and this kind of flexible connectivity could be used also in actual products. Any accurate simulations of specific fieldbused were ruled out of concept as we looked for a general solution.

We did not find any significant obstacles in implementing the functional features required by any of the mentioned applications. It is again a matter of performance to evaluate the final applicability. In the development and testing phase virtualization, timing requirements are often not as strict as in the final product. If the mentioned bottlenecks are avoided, it is possible to build almost any setup consisting of virtualized and native platforms, fast local area network and hardware bus connections. If the number of virtualized bus clients is high, we recommend using a separate virtualization server to provide enough processing power. Overall, virtual bus applicability on development and testing is very good.

Due to high performance results on native platforms using shared memory, we can safely recommend virtual fieldbus also for virtualizing systems with harder timing requirements on those platforms. Completely deterministic timings cannot however be achieved without a real time operating system. The general lack of real time guarantees is the main limiting factor in using the discussed methods for virtualization in actual products, not the virtual bus itself.

As TCP was selected one of the main technologies, extending the virtual bus system over network is possible without significant changes and enables variety of remote applications. Fieldbus based remote access, control and diagnostics without special timing requirements are thus valid applications even when connecting from outside the local network. Instead, direct applicability of a solution similar to ours is not as good for plain tunneling between physical fieldbuses, where dedicated gateway devices will presumably give better performance.

7.4 Thesis process

The full research and implementation process took almost precisely one year, from May 2012 to May 2013. This was some months longer than originally planned, but understandable in the corporate environment where priorities frequently change inside and between projects. Otherwise research proceeded as planned without exceeding its scope or requiring significant changes to the original content plan. Constructive comments were received from both the university and corporate mentors.

Theoretical and technical study was without a doubt laborious, since the rarity of existing research material on the subject forced a start from very basics. The theory discussion did find both prospects and issues, which motivated the rest of the research. In the search of best technologies, cross-virtualization communication and portability aspects were found more problematic than expected. To be able to state the final selections, multiple trials and errors were required.

A functional implementation and measurements performed on it played an essential role in verifying the assumptions and answering the questions placed in the early parts of the text. No overwhelming difficulties were found in the implementation and test phases, due to proper preparation. Based on the first results, some comparative measurements were added to the original plan to be able to fully argue the findings.

In the end, we see the thesis questions clearly and comprehensively answered, which alone proves the successfulness of the process. The conclusions were found applicable not to a single but almost any fieldbus type, which makes the information even more valuable. The supposed potential of virtual fieldbuses was proven by the positive results, which is of course a subjective success.

REFERENCES

[1] Jong-Seo, K., Sang-Hun, L. & Hyun-Wook, J. Fieldbus Virtualization for Integrated Modular Avionics. 16th IEEE Conference on Emerging Technologies and Facto-ry Automation (ETFA 2011), Toulouse, France, 5-9.9. 2011. Konkuk University, Seoul, Korea.

[2] McLoughlin, I. V. Virtualized development and testing of embedded computing clusters. Second International Conference on Networking and Computing, Seoul, Korea, 11-12.5.2011. Nanyang Technological University, Singapore.

[3] Obermaisser, R. & Peti, P. Comparison of the Temporal Performance of Physical and Virtual CAN Networks. Proceedings of the IEE E International Symposium on Industrial Electronics, Dubrovnik, Croatia, 20-23.6.2005, Vienna University of Tech-nology, Austria, pp. 1415 – 1422.

[4] IEC 61158. Digital data communications for measurement and control – Fieldbus for use in industrial control systems. International Electrotechnical Commission, 2007.

[5] Kärjä, V. Implementing Embedded Field Bus Solution into Low Power AC Drive's Control Board. Dissertation. Helsinki 2009. Metropolia University of Ap-plied Sciences. 90p.

[6] Fieldbus Inc. IEC61158 Technology Comparison [PDF]. [accessed on 16.5.2013]. Available at: www.fieldbusinc.com/downloads/fieldbus_comparison.pdf

[7] Oxford Dictionaries Online: "Virtualize" [WWW]. Oxford University Press, 2013. [accessed 16.5.2013]. Available at: http://oxforddictionaries.com/definition/english/virtualize

[8] Wilamowski, B. M. & Irwin, J. D. The Industrial Electronics Handbook: Industrial Communications Systems. Second Edition. USA 2011, CRC Press. 962p.

[9] PROFIBUS International. PROFIBUS Specification, Edition 1.0 [PDF]. 1998. [accessed on 18.5.2013]. Available at: http://www.kuebler.com/PDFs/Feldbus_Multiturn/specification_DP.pdf

[10] Corrigan, S. Controller Area Network Physical Layer Requirements [PDF]. Application report. Texas Instruments 2008. [accessed on 16.5.2013]. Available at: http://www.ti.com/lit/an/slla270/slla270.pdf

[11] Echelon. Introduction to the LonWorks Platform, Revision 2 [PDF]. USA. [accessed on 16.5.2013]. Available at: http://www.echelon.com/support/documentation/manuals/general/078-0183-01B_Intro_to_LonWorks_Rev_2.pdf

[12] EtherCAT Teghnology Group. EtherCAT – the Ethernet fieldbus [PDF]. [accessed on 16.5.2013]. Available at: http://www.ethercat.org/pdf/ethercat_e.pdf

[13] PROFIBUS and PROFINET International. PROFINET - the leading Industrial Ethernet Standard [WWW]. [accessed 16.5.2013]. Available at: http://www.profibus.com/technology/profinet/overview/

[14] Davoli, R. VDE: Virtual Distributed Ethernet. Proceedings of the First International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TRIDENTCOM'05), Trento, Italy, February 23-25, 2005. pp. 213-220.

[15] KVM - Setting guest network [WWW]. [accessed 16.5.2013]. Available at: http://www.linux-kvm.org/page/Networking

[16] Robert Bosch GmbH. CAN Specification 2.0 [PDF]. 1991. [accessed on 17.5.2012] Available at: http://esd.cs.ucr.edu/webres/can20.pdf

[17] IXXAT Automation GmbH. CANopen Basics – Introduction [WWW]. [accessed on 16.5.2013]. Available at: http://www.canopensolutions.com/english/about_canopen/about_canopen.shtml

[18] Clarke, G. & Reynders, D. Practical Modern SCADA Protocols: DNP3, 60870.5 and Related Systems. Greaat Britain 2004, Newnes. 544p.

[19] The Modbus Organization. Modbus application protocol specification V1.1b [PDF]. 28.12.2006. [accessed on 17.5.2013]. Available at: http://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b.pdf

[20] Real Time Automation, Inc. Modbus Unplugged – An introduction to Modbus [WWW]. [accessed on 16.5.2013]. Available at: http://www.rtaautomation.com/modbus/

[21] Dykstra, P. Gigabit Ethernet Jumbo Frames – And why you should care [WWW]. WareOnEarth Communications, Inc. 20.12.1999. [accessed on 16.5.2013]. Available at: http://sd.wareonearth.com/~phil/jumbo.html

[22] Digital Bond, Inc. PROFIBUS/PROFINET [WWW]. [accessed on 16.5.2013]. Available at: http://www.digitalbond.com/scadapedia/protocols/profibusprofinet/

[23] Kirrmann, H. Industrial Communication Systems: Field bus: standards [WWW]. ABB Research Center, Baden, Switzerland. 2005. [accessed on 16.5.2013]. Available at: http://www.scribd.com/doc/95424013/18/Interbus-S-4-Analysis

[24] Sampath, P. & Rao, R. E?cient embedded software development using QEMU. Eleventh Real-Time Linux Workshop, Dresden, Germany, September 20-30, 2009. ABB Corporate Research.

[25] Liebezeit, T., Junghanns, A., Bonin, M. & Serway, R. Software-in-the-Loop using virtual CAN buses: Current solutions and challenges. 5th Conference Simulation and Testing for Automotive Electronics, 10. - 11.5.2012, Berlin, Germany

[26] National Instruments. What is NI TestStand [WWW]. [accessed on 19.9.2012] Available at: http://sine.ni.com/np/app/flex/p/docid/nav-94/lang/fi/fmid/883/

[27] Smart, J. F. Jenkins: The Definitive Guide. USA 2011, O'Reilly Media.

[28] Vision Systems. NET-CAN 110 - 1 port Ethernet to CAN Bus Adapter [WWW]. [accessed 16.5.2013]. Available at: http://www.vscom.de/422.htm

[29] Esd electronics, inc. EtherCAN/2 - CAN-Ethernet-Gateway [WWW]. [accessed on 16.5.2013]. Available at: http://www.esd-electronics-usa.com/CAN-CANopen-ETHERNET-Gateway-ARM9-microcontroller-CANbus-diagnostics-SNMP-support-EtherCAN/2.html[

[30] PHYTEC America, LLC. SYS TEC CAN-Ethernet Gateway [WWW]. [accessed on 2.9.2012]. Available at: http://www.phytec.com/products/can/pc-can-interfaces/CAN-Ethernet-Gateway.html

[31] Esd electronics, inc. CAN-API Part 1: Function Description Rev 3.0 [PDF]. [accessed on 16.5.2013]. Available at: http://www.esd-electronics-usa.com/shared/handbooks/CAN-API_Part1_Function_Manual.pdf

[32] Vision Systems. VSCAN Manual [PDF]. November 2012 edition. [accessed on 16.5.2013]. Available at: http://ftp.vscom.de/multiio/others/info/VSCAN_Manual.pdf

[33] SYS TEC electronic GmbH. CAN-Ethernet-Gateway - System Manual [PDF]. 2010. [accessed on 2.9.2012]. Available at: http://www.phytec.com/pdf/manuals/L-1032e.pdf

[34] Kvaser AB. Kvaser CANlib SDK [WWW]. [accessed on 16.5.2013] Available at: http://www.kvaser.com/zh/developer/canlib.html

[35] De Niz, D., Bhatia, G. & Rajkumar, R. Model-Based Development of Embedded Systems: The SysWeaver Approach. Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium, June 4-7, 2006, San Jose, California, USA. pp. 231-242 .

[36] The MathWorks, Inc. Wärtsilä Accelerates Engine Control Development Using Production Code Generation [PDF]. 2005. [accessed on 16.5.2013]. Available at: http://www.mathworks.com/tagteam/29386_91357v00_Wartsila_us.pdf

[37] The MathWorks, Inc. Send and receive messages over CAN bus [WWW]. [accessed on 16.5.2013]. Available at: http://www.mathworks.se/discovery/can-network.html

[38] Salcianu, M., Fosalau, C. & Hariton, A. A new controller area network simulator for a rain sensing system using LabVIEW. Bulletin of the Polytechnic Institute of Jassy, LVII(2011)5, pp. 69-78.

[39] The MathWorks, Inc. Vehicle Network Toolbox - Periodic CAN Communication [WWW]. [accessed on 18.11.2012]. Available at: http://www.mathworks.se/help/vnt/examples/periodic-can-communication.html?prodcode=VN&language=en

[40] Vector. Product Information – CANalyzer [PDF]. Version 1.1. 2013. [accessed on 16.5.2013] Available at: https://www.vector.com/portal/medien/cmc/info/CANalyzer_ProductInformation_EN.pdf

[41] IXXAT Automation GmbH. VCI V3.5.1 Release [WWW]. [accessed on 16.5.2013]. Available at: http://www.ixxat.com/download_vci_v3_en.html

[42] OCERA. LinCAN (CAN bus driver) [WWW]. 2010. [accessed on 16.5.2013]. Available at: http://ortcan.sourceforge.net/lincan/

[43] Kleine-Budde, M. SocketCAN - The official CAN API of the Linux kernel. Proceedings of the 13th iCC, Hambach Castle, Germany, 2012. Pengutronix.

[44] OCERA. LibVCA (OrtCAN Virtual/Versatile CAN API library) [WWW]. 2011. [accessed on 16.5.2013]. Available at: http://ortcan.sourceforge.net/vca/

[45] Tan, V. V., Dae-Seung, Y., Myung-Kuyn, K. & Myeong-Jae, Y. Hard and Soft Real Time Based on Switched Ethernet. The 1st International Forum on Strategic Tech-nology, 18-20.10.2006, Ulsan, Korea. School of Computer Engineering and Information Technology, University of Ulsan, Korea.

[46] Chipounov, V. & Candea, G. Dynamically Translating x86 to LLVM using QEMU [PDF]. École Polytechnique Fédérale de Lausanne (EPFL), Switzerland. 2010. [accessed 16.5.2013]. Available at: http://infoscience.epfl.ch/record/149975/files/x86-llvm-translator-chipounov_2.pdf

[47] Sagar, P. M. Embedded Operating Systems for Real-Time Applications [PDF]. Seminar report. Indian Institute of Technology, Bombay, 2002. [accessed 16.5.2013]. Available at: http://perso.citi.insa-lyon.fr/afraboul/master/rtos_in_embedded.pdf

[48] Macdonell, A. C. Shared-Memory Optimizations for Virtual Machines. Doctor of Philosophy thesis. Edmonton, Alberta 2011. University of Alberta. 126p.

[49] [VBUSR] Oracle Corporation. Oracle VM VirtualBox User Manual [PDF]. [accessed on 16.5.2013]. Available at: http://dlc.sun.com.edgesuite.net/virtualbox/4.2.4/UserManual.pdf

[50] Fluke Corporation. Verifying CAN bus signals with a Fluke ScopeMeter® 120 Series [PDF]. [accessed on 25.9.2012]. Available at http://support.fluke.com/find-sales/download/asset/2392165_6003_eng_a_w.pdf

[51] Boggs, D.R., Mogul, J.C. & Kent, C. A. Measured Capacity of an Ethernet: Myths and Reality. Proceedings of the SIGCOMM ’88 Symposium on Communications Architectures and Protocols, 1988, ACM SIGCOMM, Stanford, California. Western Research Laboratory.

[52] Shochand, J. F. & Hupp, J. A. Measured Performance of an Ethernet Local Network. Communications of the ACM, 23(1980)12, pp.711-721.

[53] Jones, R. Netperf Homepage [WWW]. [accessed on 16.5.2013]. Available at: http://www.netperf.org/netperf/

[54] Iperf Homepage [WWW]. Sourceforge. [accessed on 16.5.2013]. Available at: http://sourceforge.net/projects/iperf/

[55] Gawande, T. A. & Mhala, N.N. Improve Performance Of Aodv Protocol In Wireless Ad-Hoc Network By Network Coding. IOSR Journal of Electrical and Elec-tronics Engineering (IOSRJEEE), 1(2012)5, pp. 46-51.

[56] CiscoNET. Internet Speed Issue - Bandwidth VS. Throughput [WWW]. 2009. [accessed on 16.5.2013]. Available at: http://cisconet.com/traffic-analysis/traffic-analysis-general/239-internet-speed-issue-throughput-vs-bandwidth.html

[57] Cho, K. Traffic Measurement and Analysis [PDF]. JAIST/Sony Computer Science Labs, Inc. 2003. [accessed on 16.5.2013]. Available at: http://www.sonycsl.co.jp/~kjc/papers/jaist0311-lec2.pdf

[58] Verma, D. C., Zhang, H. & Ferrari, D. Delay Jitter Control for Real-Time Communication in a Packet Switching Network. Proceedings of the TRICOM Conference, 1991. University of California at Berkeley & International Computer Science Institute.

[59] Kavi, K., Akl, R. & Hurson, a. Real-Time Systems: An Introduction and the State-of-the-Art. In: Wah, B. W. (ed.). Encyclopedia of Computer Science and Engi-neering, Vol. 4, 2009, Wiley-Interscience. pp. 2369-2377.

[60] QLogic Corporation. Introduction to Ethernet Latency: An Explanation of Latency and Latency Measurement [WWW]. [accessed 16.5.2013]. Available at: http://www.qlogic.com/Resources/Documents/TechnologyBriefs/Adapters/Tech_Brief_Introduction_to_Ethernet_Latency.pdf

[61] Cavanaugh, J. D. Protocol Overhead in IP/ATM Networks [PDF]. Minnesota Supercomputer Center, Inc. 1994. [accessed on 16.5.2013]. Available at: http://www.sonic.net/support/docs/ip-atm.overhead.pdf

[62] Hay, R. IP Packet Overhead [WWW]. [accessed 16.5.2013]. Available at: http://www.tamos.net/~rhay/overhead/ip-packet-overhead.htm

[63] Kassner, M. Noise and jitter: Nemesis of digital communications [WWW]. TechRebuplic. 2008. [accessed on 4.11.2012]. Available at: http://www.techrepublic.com/blog/networking/noise-and-jitter-nemesis-of-digital-communications/589

[64] Bovet, D. P., Cesati, M. Understanding the Linux Kernel, 3rd Edition. 2005, O'Reilly Media. 944 p.

[65] Braithwaite, K. Custom Performance Analysis using the Microsoft Performance Data Helper [WWW]. IBM WebSphere Developer Technical Journal. 2003. [accessed on 16.5.2013]. Available at: http://www.ibm.com/developerworks/websphere/techjournal/0310_braithwaite/braithwaite.html 4.11.2012

[66] Nambiar, M., Samudrala, S., Narayanan, S. Experiences with UNIX IPC for Low Latency Messaging Solutions. Proceedings of the Computer Measurement Group's International Conference, 2009. Tata Consultancy Services.

[67] Gember, A. Real-Time TCP for Embedded Devices [WWW]. Marquette University, Dept. of Mathematics, Statistics, and Computer Science. [accessed on 16.5.2013]. Available at: http://src.acm.org/gember/gember.html

[68] Schulzrinne, H., Casner, S., Frederick, R. & Jacobson, V. RTP: A Transport Protocol for Real-Time Applications. RFC 1889, IETF Network Working Group, 1996.

[69] Bova, T. & Krivoruchka, T. Reliable UDP Protocol [PDF]. 1999. [accessed on 16.5.2013]. Available at: http://www.javvin.com/protocol/reliable-udp.pdf

[70] Iqbal, A. SCTP Performance Tests [WWW]. CERN. 2003. [accessed on 16.5.2013]. Available at: http://datatag.web.cern.ch/datatag/WP3/sctp/tests.htm

[71] Österdahl, H. A comparison of TCP and SCTP performance using the HTTP protocol [PDF]. [accessed on 18.5.2013]. Available at: http://www.it.kth.se/courses/2G1305/Sample-papers/Henrik_Oesterdahl-sctp-20050525.pdf

[72] Harcsik, P., Petlund, A., Griwodz, C. & Halvorsen, P. Latency Evaluation of Networking Mechanisms for Game Traffic. NetGames '07: 6th Annual Workshop on Network and Systems Support for Games, Melbourne, Australia, September 19-20, 2007.pp. 129-134

[73] Afzal, M. K., Cantt, W., Aman-Ullah-Khan, Pescape, A., Bin Zikria, Y. & Loreto, S. SCTP vs. TCP: Delay and Packet Loss. IEEE International Multitopic Con-ference, Lahore, Pakistan, Dec 28-30, 2007.

[74] Rajamani, R., Kumar, S. & Gupta, N. SCTP versus TCP: Comparing the Performance of Transport Protocols for Web Traffic. University of Wisconsin-Madison, 2002.

[75] Pan, M. Real time communications over UDP protocol [WWW]. Code Pro-ject. 2011. [accessed on 16.5.2013]. Available at: http://www.codeproject.com/Articles/275715/Real-time-communications-over-UDP-protocol-UDP-RT

[76] Mogul, J. C. & Minshall, G. Rethinking the TCP Nagle Algorithm. ACM SIGCOMM Computer Communication Review, 31(2001) 1, pp. 6 – 20.

[77] Hacker, T. J. & Athey, B. D. The End-to -End Performance Effects of Parallel TCP Sockets on a Lossy Wide-Area Network. IPDPS '02: Proceedings of the 16th International Parallel and Distributed Processing Symposium, Fort Lauderdale, Florida, USA, 15-19 April 2002.

[78] Natarajan, P., Baker, F. & Amer, P. D. Multiple TCP Connections Improve HTTP Throughput - Myth or Fact [PDF]? University of Delaware & Cisco Systems Inc, 2008. [accessed on 18.5.2013]. Available at: http://www.techrepublic.com/whitepapers/multiple-tcp-connections-improve-http-throughput-myth-or-fact/1726301

[79] Dukkipati, N., Refice, T., Cheng, Y., Chu, J., Sutin, N., Agarwal, A., Herbert, T. & Jain, A. An Argument for Increasing TCP's Initial Congestion Window. ACM SIGCOMM Computer Communication Review 40(2010)3, pp. 26-33.

[80] Perkins, C. IP Encapsulation within IP. RFC 2004, IETF Network Working Group, 1996.

[81] Perkins, C. Minimal Encapsulation within IP. RFC 2004, IETF Network Working Group, 1996.

[82] Analytica GmbH. Manual: AnaGate TCP/IP communication [PDF]. 15.5.2009. 82p. [accessed on 17.5.2013]. Available at: http://www.anagate.de/download/Manual-AnaGate-TCPIP-V1.2.6-EN.pdf

[83] Hall, B. Beej's Guide to Unix IPC [WWW]. Version 1.1.2, 2010. [accessed on 16.5.2013]. Available at: http://beej.us/guide/bgipc/output/html/singlepage/bgipc.html

[84] MSDN. Intrprocess Communications [WWW]. Microsoft. 2012. [accessed on 16.5.2013]. Available at: http://msdn.microsoft.com/en-us/library/windows/desktop/aa365574(v=vs.85).aspx

[85] QEMU Main Page [WWW]. [accessed on 18.5.2013]. Available at: http://wiki.qemu.org/Main_Page

[86] QEMU Internals [WWW]. [accessed on 18.5.2013]. Available at: http://qemu.weilnetz.de/qemu-tech.html

[87] Coviello, G. VMShm, a mechanism for accessing POSIX shared memory from QEMU/kvm guests [WWW]. 2010. [accessed on 18.5.2013]. Available at: https://coviello.wordpress.com/2010/11/16/vmshm-a-mechanism-for-accessing-to-posix-shared-memory-from-qemu-kvm-guests/

[88] Williamson, A. VFIO: PCI device assignment breaks free of KVM. KVM Forum 2011, Vancouver, Canada, 15-16.8. 2011

[89] Libvirt Virtualization API: Virtio [WWW]. [accessed on 17.5.2013]. Available at: http://wiki.libvirt.org/page/Virtio

[90] Wagner, M. KVM Performance Improvements and Optimizations. KVM Forum 2011, Edinburgh, UK, October 21-23, 2013.

[91] Oracle Corporation. About VirtualBox [WWW]. [accessed on 17.5.2013]. Available at: https://www.virtualbox.org/wiki/VirtualBox

[92] Oracle Corporation. VirtualBox and open source [WWW]. [accessed 17.5.2013]. Available at: https://www.virtualbox.org/wiki/Editions

[93] Kurakin, A. Host-Guest communication: Bird’s Eye View [WWW]. CodeProject. 2010. [accessed on 17.5.2013]. Available at: http://www.codeproject.com/Articles/86216/Host-Guest-communication-Bird-s-Eye-View

[94] CFos software. Ping utility hrPING [WWW]. [accessed on 17.5.2013]. Available at: http://www.cfos.de/en/ping/ping.htm

[95] Fish, S. Cross-Platform Abstraction Libraries [WWW]. 2011. [accessed on 17.5.2013] Available at: http://www.shlomifish.org/open-source/portability-libs/

[96] Saks, D. The yin and yang of dynamic allocation [WWW]. 2008. [accessed on 17.5.2013]. Available at: http://www.embedded.com/design/prototyping-and-development/4007569/The-yin-and-yang-of-dynamic-allocation

[97] WDC research. What languages do you use to develop software [WWW]? 2010. [accessed on 17.5.2013]. Available at: http://blog.vdcresearch.com/embedded_sw/2010/09/what-languages-do-you-use-to-develop-software.html

[98] Oracle Corporation. Programming Interfaces Guide: POSIX Interprocess Communication [WWW]. [accessed on 17.5.2013]. Available at: http://docs.oracle.com/cd/E19963-01/html/821-1602/svipc-posixipc.html

[99] Linux man pages [WWW]. [accessed on 17.5.2013]. Available at: http://linux.die.net/man/

[100] MSDN. Desktop App Development Documentation. Microsoft. [accessed on 17.5.2013]. Available at: http://msdn.microsoft.com/en-us/library/windows/desktop/hh447209(v=vs.85).aspx

[101] Linux Programmer's Manual: Socket [WWW]. 2004. [accessed on 17.5.2013]. Available at: http://unixhelp.ed.ac.uk/CGI/man-cgi?socket+7

[102] Young, W. Winsock Programmer's FAQ: BSD Sockets Compatibility [WWW]. 2010. [accessed 17.5.2013]. Available at: http://tangentsoft.net/wskfaq/articles/bsd-compatibility.html

[103] MSDN. Porting Socket Applications to Winsock. Microsoft. 2012. [ac-cessed on 17.5.2013]. Available at: http://msdn.microsoft.com/en-us/library/ms740096%28VS.85%29.aspx

[104] Shoumikhin, A. The Differences Between Network Calls in Windows and Linux [WWW]. CodeProject. 2010. [accessed on 17.5.2013]. Available at: http://www.codeproject.com/Articles/140533/The-Differences-Between-Network-Calls-in-Windows-a

[105] O'Steen, P. Transitioning from UNIX to Windows Socket Programming [PDF]. [accessed on 17.5.2013]. Available at: http://cs.baylor.edu/~donahoo/practical/CSockets/WindowsSockets.pdf

[106] Norton, J. Dynamic Class Loading for C++ on Linux. Linux Journal 2000(2000)73es, article 38. Available at: http://www.linuxjournal.com/article/3687

[107] MSDN. Dllexport, dllimport [WWW]. Microsoft. [accessed on 16.5.2013]. Available at: http://msdn.microsoft.com/en-us/library/3y1sfaz2(v=vs.80).aspx

[108] Transmission Zero. Building Windows DLLs with MinGW [WWW]. [accessed on 16.5.2013]. Available at: http://www.transmissionzero.co.uk/computing/building-dlls-with-mingw/

[109] CodingFreak. Creating and using shared libraries in Linux [WWW]. 2009. [accessed on 17.5.2013]. Available at: http://codingfreak.blogspot.com/2009/12/creating-and-using-shared-libraries-in.html

[110] CMake Homepage [WWW]. [accessed 17.5.2013]. Available at: http://www.cmake.org/

[111] Maritime connector. World's first LNG powered passenger ship handed over to Viking Line [WWW]. 11.1.2013. [accessed on 17.5.2013]. Available at: http://maritime-connector.com/news/general/world-s-first-lng-powered-passenger-ship-handed-over-to-viking-line/

[112] Pietilä, M. World's largest engine-driven power plant for the Middle East. Twentyfour7 – Wärtsilä Stakeholder Magazine 2013(2013)1, p. 60-61.

[113] Wärtsilä 64: The world's most powerful medium-speed engine [WWW]. [accessed on 17.5.2013] Available at: http://www.wartsila.com/fi/engines/medium-speed-engines/wartsila64

[114] Pensar, J. & Storbacka, M. UNIC – The reliable solution for robust industrial controls. Wärtsilä Technical Journal 2007(2007)2, p. 40-44.

[115] Wärtsilä Medium-speed Engines [WWW]. [accessed on 17.5.2013]. Available at: http://www.wartsila.com/file/Wartsila/fi/1278529609886a1267106724867-Wartsila-O-E-W-MS.pdf

[116] About Wapice [WWW]. [accessed on 17.5.2013]. Available at: http://w3.wapice.com/en/about-us/wapice

[117] Junnila, S., Pajula, R., Shroff, M., Siurulainen, T., Kwitek, M., & Tuominen, P. Design of High-Performance CAN Driver Architecture for Embedded Linux. Pro-ceedings of the 13th iCC, Hambach Castle, Germany, 2012. Wapice Ltd.

[118] Valkola, L. Test application for CAN-devices. Dissertation. Vaasa 2009. Vaasa University of Applied Sciences. 55p.